# PC-8201
# $N_{82}$-BASIC
## REFERENCE MANUAL

NEC

PC-8201-RM
PTS-127

# PC-8201

# $N_{82}$-BASIC

## REFERENCE MANUAL

# まえがき

本書はPC-8201をBASICモードで使用する時のプログラミング言語，$N_{82}$-BASICについての解説書です．BASICを親しみやすく解説するためにサンプルプログラムなどをできるだけ工夫してあります．更に実際のプログラミングの参考としていただくために，第5章，第6章などを特に設けてあります．初心者の方も短いサンプルプログラムなどを打ち込んで，BASICに慣れ親しんでみてください．

# マニュアルの見方

本書は$N_{82}$-BASICの機能，命令などをわかかりやすく解説するために次のような構成になっています．

### 第1章　$N_{82}$-BASICの概要

$N_{82}$-BASICの特徴や機能を，コマンド・ステートメントなどの説明に先だって解説しています．その内容は，BASICを正しく使う上での最低限の知識と，$N_{82}$-BASICで追加されている特殊な機能(特にファイル管理)の紹介や，それを使う上での知識です．これらは，第2章，第3章の文法の説明を理解するために必要な事項です．必ず読むようにしてください．ただしBASICを初めて学ぶ方で，これらを読むのが重荷であると感じられる場合は，ユーザーズマニュアルの第4章や本書のサンプルプログラムなどを使って，とりあえずBASICに慣れてしまってからでも遅くはありません．

### 第2章　コマンド・ステートメント

### 第3章　関数

BASICの機能は大きく分けるとコマンド・ステートメントと関数に分類することができます．

コマンド及びステートメントはBASICがもつ命令です．コマンドは一般にダイレクトモードで使用するもので実行後〝Ok〟が表示され，ステートメントは主にプログラム中で使用しますが明確に分かれているわけではありません．

関数は常に何らかの命令に付随して引用される機能で，単独で用いることはできません．

見出し語のところに（ディスク），または（CRT）とあるものは，それらの機器が接続されていなければ使えない専用命令及び関数です．

### 第4章　機械語プログラム及びキャラクタ定義について

$N_{82}$-BASICで用意されている機械語プログラムとのリンク機能及びキャラクタ定義機能について解説してあります．

### 第5章　サンプルプログラミング

$N_{82}$-BASICによるプログラミングを実例を通して紹介します．

**第6章　プログラミングの問題と対策**

自作のプログラムが思い通りに働かない，エラーがでるという場合の原因と対策についての章です．

**第7章　資料**

キャラクタコード表，エラーコード表などプログラミングに必要な資料から構成されています．

**索引**

項目や用語による索引です．

本書のBASICの文法説明などにおいて，次のような事柄を定義してあります．

**機　能**　命令の機能を簡単に示します．

**書　式**　命令の記述の仕方を示します．実際の入力時には次のような決まりに従ってください．

1．アルファベットの大文字で示された項目は，そのまま入力します．入力する場合は，小文字でも大文字でもかまいません．ただし，引用符（"）で囲まれた文字列（ファイル名など）は，大文字と小文字を区別してください．
2．カギカッコ〝〈 〉〟で囲まれた項目は，ユーザーが必ず指定します．
3．角カッコ〝〔 〕〟で囲まれた項目は，オプションであり省略することができます．省略した場合，デフォルト値（BASICによって自動設定される値）または以前に指定した値が適用されます．
4．上記のカギカッコ，角カッコ以外の記号でカッコ〝( )〟，カンマ〝,〟，セミコロン〝;〟，ハイフン〝－〟，等号〝＝〟などの記号は示された位置に正しく入力します．ハイフン〝－〟はキャラクタとしてはマイナス〝－〟と区別されていません．
5．省略記号〝…〟の続く項目は，1行の許す長さ（254文字）以内で任意の回数繰り返すことができます．
6．本文中で1つの記号や文字を意味しているとき，本文そのものとの混乱を避けるために引用符で囲ってありますが，引用符そのものを意味するときだけはカッコで囲んであります．

例）本文中で

… このハイフン〝－〟は …

・・・その引用符（"）の後に・・・

のように書いてある場合，まん中の記号あるいは文字一つのことを意味しています．

7. 書式の中でカギカッコ〝＜＞〟の内容をパラメータと呼び，大きくわけて3種類あります．

①行番号　必ず数字だけを使い，式や変数は使えません．

②文字列　特にことわりがない限り引用符で囲まれた文字列，文字型変数，両者の組み合わせでできた文字式のどれでも使うことができます．

③数値　特にことわりがない限り，定数，数値，変数，数式（関数や論理式も含む）のどれでも使うことができますが，その数値の範囲はそれぞれ限界があります．また整数を指定すべき所に実数を指定した場合などは型変換が行われます（第1章1.10参照）．

8. キーワードとパラメータ，カンマなどの間の空白（スペース）は，あってもなくてもかまいません．

**文　例**　実際の入力の仕方の見方として簡単な例を示します．

**解　説**　命令の使用方法や詳しい機能とそれに関して注意しなければならない点などを説明します．

**注　意**　命令の使用上，特に間違いやすい点を注意してあります．

**参　照**　その命令に関連する，他の命令や項目を掲げてあります．

**サンプルプログラム**　いくつかの文を交えたプログラム例を示します．

# 目 次

# N82-BASICの概要

# 1 $N_{82}$-BASICの概要

## 1.1 $N_{82}$-BASIC

$N_{82}$-BASICは，PC-8201の多彩な機能を有効に引き出すために開発されたプログラミング言語です．基本的にはPCシリーズを操作するN-BASIC，$N_{60}$-BASIC，$N_{88}$-BASICと互換性を持っていますが，ハードウェアの違いなどによって異っている部分もあります．最も近いのはN-BASICで画面操作や機械語制御関係の命令の一部を除けば，命令や関数の書式も同じになっています．またN-BASICとは中間言語レベルでの互換性をもっているため，N-BASICで書かれたプログラムはASCII形式でセーブしてないものでも直接ロードすることができます．

## 1.2 $N_{82}$-BASICの特徴

$N_{82}$-BASICは新しい機能として次のような特徴を持っています．

1. PC-8201が持っているハードウェアの機能のほとんどを$N_{82}$-BASICで使用できます．

   内部
   ○プログラマブル・ファンクションキー
   ○リアルタイムタイマー
   ○ RAM上のファイル管理
   ○圧電スピーカ
   ○オート電源スイッチ
   ○ LCDディスプレイ
   外部
   ○オーディオカセット
   ○ RS-232C
   ○ミニフロッピィディスク
   ○セントロニクス仕様プリンタ
   ○バーコードリーダ
   ○外部記憶装置
   ○ ROM／RAMカートリッジ
   ○専用CRTディスプレイ

2.LCD画面が大きく，ポータブルコンピュータとしては非常に高度なグラフィックスを扱えます（240×64ドット）.
3.TEXTモードでのエディットとスクリーンエディットの二つのプログラム編集モードが使えるので，プログラムの作成，デバッグが容易に行えます.
4.ON COM GOSUBなどの割り込み操作命令により，通信回線を有効に利用することができます.
5.バンク切り換えにより，ユーザーズメモリは最大96K近くまで拡張でき，大規模なプログラムを作成したり，多くのファイルをRAM上に置くことができます(標準装備では16K RAM．バンク切り換えなし).
6.最大21個(バンク切り換えで63個)までのファイルをRAM上に置くことができ，ディスク装置を内蔵しているような感覚でファイルを操作することができます.
7.POWER命令により，電池が無駄に消耗してしまうのを防ぐことができます.
8.各種の，バックアップ機構により，電源を切っても少量の電力を使ってRAMの内容を保存するため,誤操作によるプログラムの消滅を防ぐことができます.

## 1.3 動作モード

BASICを起動すると(起動方法についてはユーザーズマニュアルを参照してください)，PC-8201は，画面に〝Ok〞という文字を出力し，BASICはコマンドレベルになります．この状態のときは，あらゆるコマンドをキーボードから入力，実行することができます．このときコマンドなどの文や行の先頭に行番号をつけてある場合は，プログラムとしてメモリの中に格納されるだけで実行はされません.

### 1.3.1 ダイクレトモード

行番号をつけずにBASICの文法に合った文を入力した場合，その文は，キャリッジリターン（[↵]キー）を入力後，すぐ実行されます．これをダイレクトモードにおける実行といいます.

### 1.3.2 プログラムモード

行番号（0～65529）を先頭につけて文を入力した場合，それはメモリの中にプログラムとして行番号とともに格納されます．そしていったん格納されたプログラムは，RUNコマンドおよびGOTO文，GOSUB文によって実行させることができます．これをプログラムモードによる実行と呼びます．プログラムをすべて実行し終るか，[STOP]キー入力やエラーの発生で実行が停止するとBASICはコマンドレベルに戻ります.

## 1.4 文

文とは，BASICが行う手続きを記述している最小単位です．

文には，BASICが実行する式，ステートメント，コマンド，関数などを書くことができます．また文は，コロン〝：〟を用いて他の文とつなぐことができます．これは複文（マルチステートメント）と呼ばれ，複文は，1行（行番号も含めて）254文字以内の長さまで許されています．

## 1.5 行番号

BASICの各プログラム行は，行番号で始まらなければなりません．行番号には，0～65529までの整数を用います．行番号はプログラム行をメモリに格納する順序を示し，実行も行番号の若い方から行われます．また分岐や編集の目印としても使用されます．

行番号の代りにピリオド〝.〟を使うことができる場合があります．ピリオドはEDIT，LIST，RENUMなどのコマンド中で，エラー発生，編集などによってBASICのポインタが示している現在の行を表すものです．

例）
```
LIST.
EDIT.
```

## 1.6 使用できる文字とコントロールキャラクタ

$N_{82}$-BASICの使用できる文字は，英文字，カナ，数字，特殊記号，そしてグラフィックキャラクタより構成されており，これらはキャラクタセットと呼ばれます．英文字は大文字と小文字で，数字は0から9まででです．これらキャラクタセットの詳細は，第7章 資料の〝キャラクタ表〟を参照してください．他にも特別の意味をもつコントロールキャラクタがあります．これについても第7章 資料を参照してください．

## 1.7 特殊記号の使い方

BASICでは，演算子（+，−，*，／）などの他にも特別な意味を持つ記号があります．ここでその意味をまとめて説明しておきます．

1.ピリオド（.）

現在BASICが着目している行番号の値を持っておりポインタとして使用すること

ができます．BASIC が着目しているとは，新しい行を挿入した，エラーが発生したなどの行です．

例）　LIST.

2.ハイフン（-）

LIST，EDIT 命令など行の範囲を指定する時，何行から何行までという場合に使います．

例）　EDIT　100-200

3.コロン（：）

マルチステートメントの区切りとして使います．

例）　A=B+C：PRINT　A

4.カンマ（,）

PRINT，INPUT，DATA などパラメータや数値が並ぶ場合その区切りとして多用されます．

例）　INPUT　A，B，C
　　　DATA 8，64，256

5.セミコロン（；）

PRINT 文などの区切りとして使います．

例）　PRINT〝A=″；A

6.アポストロフィ（'）

REM 文（リマーク）の代用として使えます．

7.疑問符（?）

PRINT 文の代用として使えます．

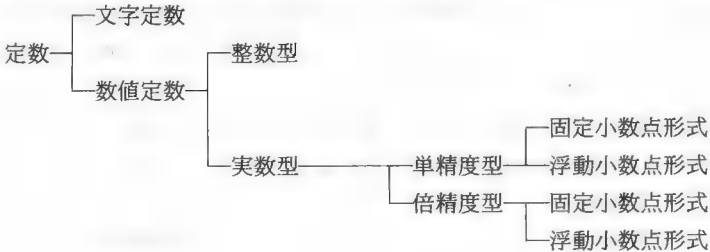
例）　?5*3.14

8.スペース

BASIC は原則としてスペースを無視します．ただしコマンド，ステートメント，関数の前にスペースを入れるとその分のメモリを使います．またコマンド，ステートメント，関数のキーワード自体にスペースを入れることは通常できません（例えば IN　PUT は不可）．

# 1.8 定数

## 1.8.1 定数の種類

N82-BASICの定数には以下のものがあります.

## 1.8.2 文字定数

文字定数とは，255文字以下の引用符（″）で囲まれた英数字，カナ文字，記号などの列のことです．なお，(″)を文字列の中に記述する場合は，CHR$関数を用いなければなりません．

例）　"This is a string."
　　　"3.1415926535897932"
　　　"ポータブルコンピュータ"

## 1.8.3 数値定数

数値定数は，整数型，実数型に分けられ，それらは正あるいは負の数，または0です．

負の数の前には必ず符号をつけなければなりませんが，正の数の前の符号は省略することができます．

## 1.8.4 整数型

－32768から＋32767までのすべての整数．または，数の後に％をつけたもの．小数点をつけることはできません．

例）　32767
　　　－123
　　　32767%←整数を表す.

### 1.8.5 実 数 型

実数型は，単精度型と倍精度型に分けられます．

### 1.8.6 単精度型

有効桁7桁の精度で格納されます．出力のときは7桁目が四捨五入され，6桁以下で表示されます．扱える数値は，－1.70141E＋38～1.70141E＋38です．

(1) 7桁以下の実数

(2) Eを使った指数形式

(3) 最後に！を伴った数

例） 1.23
　　－7.06E＋06
　　3525.68
　　3.14！

### 1.8.7 倍精度型

有効桁16桁の精度で格納され，16桁以下で表示されます．扱える数値は，－1.701411834604692 D＋38～1.701411834604692 D＋38です．

(1) 8桁以上の数

(2) Dを使った指数形式

(3) 最後に#を伴った数

例） 1234567890
　　－1.09432D＋06
　　＋0.3141592653D＋01
　　56789.0#
　　8657036.1543976

## 1.9 変数

### 1.9.1 変数とは？

BASICのプログラム中で使われる値を格納するためのエリアに，英数字から成る名前を対応させたものが変数です．

変数の値は，プログラマによって定義され，演算，参照などに使うことができます．これは演算，代入の実行後，割り当てられます．数値変数は，値が割り当て

られるまえに参照されたら0が代入され，文字型変数はヌルストリング（空の文字列）が代入されます．

### 1.9.2 変数名と型宣言文字

$N_{82}$-BASICにおいて，変数名は2文字までの英字で始まる英数字で判別されます．3文字以上の変数名をつけてもかまいませんが最初の2文字までの判別しかしません．また，変数名の中のカナやグラフィックキャラクタ及びスペースはまったく無視されます．

**例1）　ABC＝5，AアB＝5，A　B＝5などはすべてAB＝5と同じ．**
**A＝3，A1＝3，Aア2＝3などは区別される．**

変数名は予約語（BASICで使われるコマンド・ステートメントなどのキーワード）であったり，予約語を含んではなりません．また英文字において大文字，小文字の区別はありません(第7章 予約語表参照)．

例2）
- CT、COUNT　変数になれる
- COST　変数になれない（COSを含んでいる）

同じ変数名でも型が違えば区別されます．変数の型は型宣言文字によって決まり，型宣言文字は変数名の最後につけて，その変数の型を表します．型宣言文字を省略すると，“！”がついているとみなされます（単精度実数型変数となる）．

型宣言文字
- %　整数型
- !　単精度実数型
- #　倍数度実数型
- $　文字型

例3）
A
A#
A%
A $
これらは区別されるが，AとA！は同じ．

変数の型を宣言するものにもう一つ便利な方法があります．それは DEFINT, DEFSNG, DEFDBL, DEFSTR の型宣言文をプログラム中で用いる方法です．これについては第2章で詳しく説明します．

### 1.9.3 配列変数

配列とは，一つの変数名でいくつかの要素を参照することのできる変数です．配列変数のそれぞれの要素は，整数または整数表記による添字によって参照されます．

配列変数の次元は255次元(ただし実際には1行の長さで指定できないのでもっと低次元になります.)まで，添字はメモリの範囲内で許されており，これらの大きさはDIM文(第2章参照)で宣言します．ただし，各添字は原則として0から始まりますから，実際の要素数は添字の数＋1となります．

例) DIM A(10)　　1次元配列，要素の数＝11
DIM TA(10,50)　　2次元配列，要素の数11×51＝561
DIM NAME$(2,5,3)　　3次元配列，要素の数3×6×4＝72

注：各添字の数が10以下の時はDIMによる宣言を省略することができます．

### 1.9.4 予約変数

$N_{82}$-BASICには，BASIC自身が専用に使う予約変数があります．これらの変数は，ユーザーによって一般の変数として使うことはできません．

TIME$　　現在の時分秒をHH：MM：SSの形でもっています．同じ形で代入することもできます．

DATE$　　現在の年月日をYY／MM／DDの形でもっています．同じ形で代入することもできます．

ERL　　エラーの生じた時，エラーが発生した行番号をもっています．代入することはできません．

ERR　　エラーが発生した時，生じたエラーのエラーコードをもっています．代入することはできません．

これらについては第3章で詳しく説明してあります．

## 1.10 型変換

N82-BASICの数値データは，必要に応じてその型から他の型に変換することができます．ただし，文字型と数値型の間でこの変換を行うことはできません．

(1) ある型の数値データが，違った型の数値変数に代入された場合，数値は，その変数名によって宣言された型に変換されます．

例）
```
10  ABC%=1.234
20  PRINT  ABC%
RUN
 1
```

(2) 精度の違う数値間の演算の場合，精度の高い方に変換されて，演算が行われます．たとえば，10#／3の場合は，10#／3#として演算が行われます．

例）
```
10  A#=10#/3
20  B#=10#/3#
30  PRINT  A#, B#
RUN
3.333333333333333     3.333333333333333
```

(3) 論理演算の場合，扱われる数値はすべて整数に変換され，結果は整数で与えられます．

例）
```
10  A#=12.34
20  B=NOT  A#
30  PRINT  B, A#
RUN
-13            12.34000015258789
```

(4) 実数が整数に変換される場合は，小数点以下は切り捨てられます．この時，整数型で扱える範囲を超えた場合はエラーが起こります．

例）
```
10  A%=34.4
20  B%=34.5
30  PRINT  A%, B%
RUN
34             34
```

```
10  A#=1.234E+07
20  B%=A#
30  PRINT  B%, A#
RUN
? OV Error in 20
```

(5) 倍精度変数が単精度変数に代入された時は，変数の値は有効数字7桁に丸めたものとなります．単精度変数の精度は7桁であり，もとの倍精度の数値の8桁目以降の範囲で誤差が生じます．

例)
```
10  A#=1.23456789#
20  B!=A#
30  PRINT  A#, B!
RUN
1.23456789         1.23457
```

## 1.11 式と演算

式とは定数や変数を演算子で結合した一般的な数式や，単に文字や数値，あるいは変数だけのものをいいます．

例)
```
"BASIC"
3.14
10+3/5
A+B/C-D
TAN (DO)
```

BASIC の演算は次の5つに分類されます．

1. 算術演算
2. 関係演算
3. 論理演算
4. 関数
5. 文字列演算

### 1.11.1 算術演算

算術演算子には次のようなものがあります．

| 実行順序↓ | 演算子 | 演算 | 例 |
|---|---|---|---|
| | ∧ | 指数演算 | X∧Y |
| | − | 負号 | −X |
| | ＊，／ | 乗算，実数の除算 | X＊Y，X／Y |
| | ＋，− | 加算，減算 | X＋Y，X−Y |

演算の実行順序を変更する場合はカッコを使用します．カッコの中の演算子は

他の演算子より先に実行されます．カッコ内においては通常の実行順序に従います．

次に実行例を示します．

| | 代数表記 | BASICの表記 |
|---|---|---|
| 1) | $2X+Y$ | 2＊X＋Y |
| 2) | $\frac{X}{Y}+2$ | X／Y＋2 |
| 3) | $\frac{X+Y}{2}$ | (X＋Y)／2 |
| 4) | $X^2+2X+1$ | X∧2＋2＊X＋1 |
| 5) | $X^{Y^2}$ | X∧(Y∧2) |
| 6) | $(X^Y)^2$ | X∧Y∧2 |
| 7) | $Y(-X)$ | Y＊(－X) |

### (1)整数の除算と剰余の計算

整数の除算は¥によって行われます．扱われる数値が実数の場合は，演算が実行される前に小数点以下が切り捨てられます．商は小数点以下が切り捨てられた整数となります．

例)　10¥3＝3　　　(10／3＝3…　1)
　　 24.75¥5＝4　(24／5＝4…　4)

剰余の計算はMODによって行われます．扱われる数値が実数の場合は，演算が実行される前に小数点以下が切り捨てられます．結果は整数の割り算の余りです．

例)　13.3　MOD　4＝1　　　(13／4＝3…　1)
　　 25.68　MOD　6.99＝1　(25／6＝4…　1)

### (2) 0での除算

式の実行時に0での除算(／，¥，MOD)が行われた場合は，エラーメッセージを出力し，コマンドレベルに戻ります．

また，べき乗の実行時に，0に対して負のべき乗を行った場合も同様になります．

例)　PRINT 2／0
　　 ?／0 Error

```
PRINT 0∧－1
?／0 Error
```

**(3)桁あふれ（オーバーフロー）**

代入や演算の結果がその変数の型内で表現することのできる範囲をこえた場合，桁あふれが発生します．

桁あふれが起こった場合，BASIC は〝?OV Error″（Overflow エラー）を出力し，コマンドレベルに戻ります．

例）
```
PRINT 3∧300
?OV Error
```

## 1.11.2 関係演算

関係演算子は 2 つの数値を比較するときに用います．結果は，真(－1)，偽(0)で得られ，条件判定文などプログラムの流れを変えるのに用いられます（IF 文参照）．

| 関係演算子 | 内容 | 例 |
|---|---|---|
| ＝ | 等しい | X＝Y |
| ＜＞，＞＜ | 等しくない | X＜＞Y，X＞＜Y |
| ＜ | 小さい | X＜Y |
| ＞ | 大きい | X＞Y |
| ＜＝，＝＜ | 小さいか等しい | X＜＝Y，X＝＜Y |
| ＞＝，＝＞ | 大きいか等しい | X＞＝Y，X＝＞Y |

**注意**：＝は代入文にも使うので注意すること．

IF 文の中での使い方の例を次に示します．
```
IF  X=0  THEN 1000
IF  A+B<>0  THEN  X=X+1 : Y=Y+1
```

## 1.11.3 論理演算

論理演算子は複数の条件を調べたり，ビット操作やブール演算を行ったりするのに用います．論理演算はビットごとに 0 または 1 を結果として与えます．

各論理演算の内容を次に示します.

NOT = not（否定）

| X | NOT X |
|---|---|
| 1 | 0 |
| 0 | 1 |

AND = and（論理積）

| X | Y | X AND Y |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

OR = inclusive or（論理和）

| X | Y | X OR Y |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

XOR = exclusive or（排他的論理和）

| X | Y | X XOR Y |
|---|---|---|
| 1 | 1 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

IMP = implication（包含）

| X | Y | X IMP Y |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 1 |
| 0 | 0 | 1 |

EQV = equivalence（同値）

| X | Y | X EQV Y |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |

論理演算子も関係演算子のように，プログラムの流れを変えるのに用いられます．この場合，論理演算子は２つ以上の関係演算子を結ぶことができます．

例）
(1) IF X＜0 OR 99＜X THEN 1000
(2) IF 0＜X AND X＜100 THEN X＝0
(3) IF NOT（A＝0）THEN 20

(1) Xが負または，99より大きければ，行番号1000へ飛ぶ．
(2) Xが正でかつ，100より小さければ，Xに０を代入する．
(3) Aが０でなければ，行番号20へ飛ぶ．

**注意**：論理演算子は演算の前に扱う数値を－32768から＋32767までの２の補数表示の整数に変換します．もし，この範囲外となれば“Overflow”（OV Error）エラーとなります．もし，０（偽）と－１（真）しか与えられなかったら，論理演算子は０と－１しか結果として与えません．

指定された論理演算では，この整数に対しビットごとに演算を行います．したがって，論理演算子はバイトデータをあるビットパターンに照らし合わせて調べることができます．

たとえば，AND演算子はＩ／Ｏポートのステータスバイトの必要なビット以外のすべてのビットをマスクすることに使えます．またOR演算子はある２進数を作るために２つのビットパターンを混在させることができます．

以下の例は論理演算子がどのように働くかの例です．

63 AND 8 ＝ 8　　$63=(111111)_2$，$8=(001000)_2$
したがって，63 AND 8 ＝ $(001000)_2=8$

| 式 | 説明 |
|---|---|
| −1 AND 8 = 8 | $-1=(1111111111111111)_2$, $8=(001000)_2$<br>したがって，−1 AND 8 = 8 |
| 12 OR 11 =15 | $12=(1100)_2$, $11=(1011)_2$<br>したがって，12 OR $11=(1111)_2=15$ |
| 32767 OR −32768=−1 | $32767=(0111111111111111)_2$<br>$-32768=(1000000000000000)_2$<br>したがって，32767 OR −32768= $(1111111111111111)_2=-1$ |
| 12 XOR 11 = 7 | $12=(1100)_2$, $11=(1011)_2$<br>したがって，12 XOR $11=(0111)_2=7$ |
| 10 XOR 10 = 0 | $10=(1010)_2$<br>したがって，10 XOR $10=(0000)_2=0$ |
| NOT X =−(X+1) | 任意の数の補数表示は1の補数に1を加えたものである． |

なお，論理演算は厳密には関数ではありませんが各演算子については第3章の関数でも説明してあります．

### 1.11.4 関　数

関数とは与えられたある引数に対して，ある決った演算を行うもので，値としては，この演算の結果を返します．

BASICは〝組み込み関数〟としてSIN（正弦），SQR（平方根）などの数値関数やCHR$, LEFT$などの文字列関数を本体内にもっています．これらの関数については第3章で詳しく説明します．

また，これらの初等関数（SIN関数など）は，引数と結果をすべて単精度として計算します．

一般に引数に整数しかとらないものは，小数部分を切り捨てて整数に丸めてから演算を行います．

次にその使い方の例を示します．

```
A＝SIN (3.14)＋COS (3.14)
PRINT 2, 2＊2, SQR(2)
```

## 1.12 文字列の演算

### 1.12.1 文字列の連結

文字列は演算子〝＋〞によって連結することができます．

例）
```
10  A$＝〝NEC〞: B$＝〝PORTABLE〞: C$＝〝COMPUTER〞
20  D$＝A$＋〝 〞＋B$＋〝 〞＋C$
30  PRINT D$
RUN
NEC PORTABLE COMPUTER
```

### 1.12.2 文字列の比較

文字も数値の比較に用いられるものとまったく同じ関係演算子を用いて比較することができます．

＝, <, >, <>, ><, <＝, ＝<, >＝, ＝>

文字列の場合それぞれの文字列の最初から1文字ずつ，文字の比較を行います．もし相互にまったく同じ文字列の場合は，その2つの文字列は等しくなりますが，1箇所でも違った場合は，その文字のキャラクタコード（第7章 資料のキャラクタコード表参照）の大きい方の文字列が大きくなります．文字列の片方が短かくて比較が途中で終わった場合は，短かい文字列の方が小さくなります．

文字列の比較においては，空白なども意味をもちますから注意してください．

例）
```
   〝AA〞 < 〝AB〞
〝BASIC〞 = 〝BASIC〞
   〝X&〞 > 〝X #〞
〝PEN  〞 > 〝PEN〞
   〝cm〞 > 〝CM〞
 〝DESK〞 < 〝DESKS〞
  〝ABC〞 < 〝AD〞
```

このように，文字列の比較は文字列の内容を調べたり，文字をアルファベット順に並べたり（ソート）することに使うことができます．

## 1.13 演算の優先順位

演算は次の順序によって行われます．

1. カッコで囲まれたもの
2. 関数
3. 指数（べき乗）∧
4. 負号（－）
5. *，／
6. ¥
7. MOD
8. ＋，－
9. 関係演算子（<，>，＝など）
10. NOT
11. AND
12. OR
13. XOR
14. IMP
15. EQV

## 1.14 エラーメッセージ

$N_{82}$-BASICは，プログラムの実行を中断させなければならないようなエラーを実行時に検出した時，エラーメッセージをプリントしコマンドレベルに戻ります．

ダイレクトモードでのエラーメッセージの書式は，

```
XX
```

プログラムモードの書式は，

```
XX in lllll
```

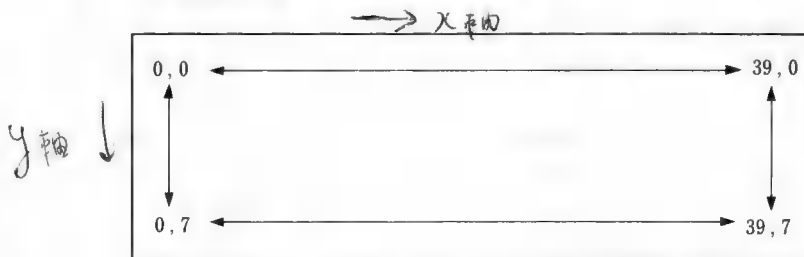
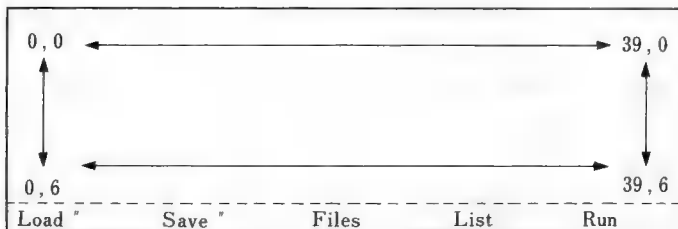
XXはエラーメッセージで，lllllはエラーが検出された行番号です．

$N_{82}$-BASICのエラーコードとエラーメッセージの内容については，第7章 資料の〝エラーコード表〟を参照してください．

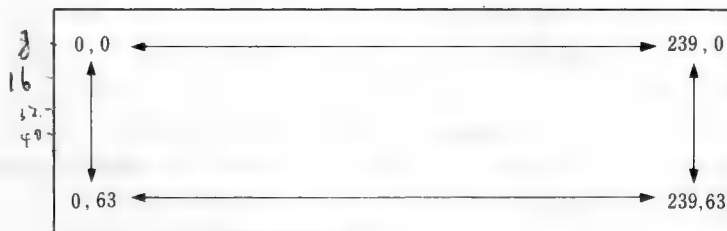
## 1.15 画面

$N_{82}$-BASIC の画面（LCD）は次のような構成になっています.

● 文字

横40×縦8文字（ただしファンクションキー表示中は縦7）. LOCATE 文で指定する座標はこれに従い，左上隅を（0，0）とし，右下隅を（39，7）とします.

テキスト画面

ファンクションキー表示中

● ドットグラフィック

横240×縦64ドット（ファンクションキー表示には左右されません）. PSET 文で指定する座標はこれに従い，左上隅を（0，0）とし，右下隅を（239，63）とします.

グラフィック画面

## 1.16 プログラムのエディット(編集)

$N_{82}$-BASICは，プログラムの作成，編集モードを二つ持っています．

### 1.16.1 スクリーンエディット

BASICがコマンドレベル（ダイレクトモードも同義）にある時，画面上の任意の行の任意の位置にカーソルを移動し，任意の文字や記号を入力して書き換えることによって編集します．編集は必ず行単位で行い，1行の追加，変更を終わったら[↵]キーを押さなければなりません．[↵]キーを押さない場合，画面上の行がどうあろうとその行には何も追加，変更は行なわれず，メモリ上は以前のままになります．

字句を挿入する場合はまず任意の位置までカーソルを移動し，[PAST INS]キーを押します．するとカーソルがアンダーカーソル（"＿"）に変わり，インサートモードになります．ここで必要なだけ挿入を行い，カーソル移動キーでカーソルを移動するかもう一度[PAST INS]キーを押すことによってインサートモードをぬけ出せます．

字句を削除する場合，任意の位置までカーソルを移動し，[DEL BS]キーを押します．するとカーソルのすぐ左の1字が消され，カーソル以下はつながって移動します．必要なら更に[DEL BS]キーを押してこの作業を操り返します．この時，[SHIFT]＋[DEL BS]キーを押すと，カーソルは移動せず，カーソル位置の文字が消されてそれ以下がつながって移動します．またカーソルを任意の位置に移動し，[CTRL]＋[E]を入力するとカーソルのある位置以後，その行の終りまでをすべて削除します．

挿入，削除共，行単位の編集であることに変わりはなく，1行の編集を終えたら必ず[↵]キーを押さなければなりません．

ある行を丸ごと削除する場合は，行番号だけを入力し[↵]を押します（画面上に既に削除したい行が表示されている場合，その行番号の右にカーソルを移動し，[CTRL]＋[E]と[↵]を入力しても同じことになりますが，行番号ごと[CTRL]＋[E]で画面から消してしまうと，削除は行われません)．

ある行をコピーする場合は，行番号の上にカーソルを移動して必要な行番号にして[↵]を押すだけで済みます．この時行番号の他に必要な部分を変更してから[↵]を押して新しい行を作ることもできます．いずれの場合も以前の行はまったく変更されずに残っているので，似たような行をいくつも書く必要がある時には便利に使えます．

## ■スクリーンエディットに使う特殊キーとコントロール文字

| キー | 機　能 |
|---|---|
| ↑↓←→ | カーソルを上下左右に一つずつ移動する |
| PAST INS | 挿入モードにする |
| DEL BS | カーソルの左の1文字を消し，カーソル以下がその位置につながって移動する |
| SHIFT ＋ DEL BS | カーソル位置の文字を消し，カーソルより後の部分がつながって一つ前へつめる |
| ↵ | 1行を入力する |
| TAB | カーソル位置から1，11，21，31(画面左隅から数えた位置)のどれかに達するまで空白を出力する |
| STOP | エディット中は作業の中止を意味する |
| CTRL ＋ C | STOP と同じ |
| CTRL ＋ E | カーソル以下を削除 |
| CTRL ＋ H | DEL BS と同じ |
| CTRL ＋ I | TAB と同じ |
| CTRL ＋ K | カーソルをホームポジション(画面左上隅)に移動する |
| CTRL ＋ L | 画面を消去し(プログラムは消えない)カーソルをホームポジションに移動する |
| CTRL ＋ M | ↵ と同じ |
| CTRL ＋ Q | CTRL ＋ S を解除する |
| CTRL ＋ R | PAST INS と同じ |
| CTRL ＋ S | プログラムをLIST中，画面のスクロールを一時停止する |
| CTRL ＋ U | 1行を画面から消去。プログラムの行は消えない |

### 1.16.2 TEXT モードのエディット

BASICから，EDITコマンドの実行によってTEXTモードに移り，プログラムの編集を行います．BASICに復帰するには[ESC]キーを2回押すか，[f･10]キーを押します．TEXTモードでは常にインサートモードになっているので，キーボードから入力した字句は，カーソルの位置へ次々と挿入されます．自動ワードラップ機能が働かないことを除いて，編集方法はすべて通常のTEXTモードと変わりありませんので，詳しくはユーザーズマニュアルの〝第3章 TEXT〟を参照してください．

TEXTモードで作成，編集されたBASICのプログラムは，表示されているものがメモリ上のプログラムとまったく同じになるため，変更，挿入，削除は画面上で行われると同時に進行します．従って，スクリーンエディットのように行単位で編集するという制約がなく，変更するごとに[↵]キーを押す必要はなくなります．TEXTモードでの[↵]は，「改行」を意味し，必要な箇所には〝↲〟が既に表示されています（新たな行を作る場合にはやはり[↵]キーを入力しないと前後の行とつながってしまいます）．

またTEXTモードで編集した結果，行番号のない文や，行番号だけの行ができてしまうと，BASICのプログラムとしての形が失われ，BASICに復帰できなくなることがあります．こういう場合はブザーの音と共に〝Text ill-formed〟と表示され，間違っている行がカーソルで示されます．正しい形に直せば復帰できます．

TEXTモードでの編集を利用すると，スクリーンエディットでの編集よりも，もっと大がかりな編集を行うことができます．例えばPASTEバッファを利用すると，あるプログラムの一部だけを抜き出し，他のプログラムに利用したり，同じプログラムの中でも，よく似た部分をそっくり他の部分にコピーしたりする時などは，スクリーンエディットよりも簡単に行うことができます．また，ある特定の変数名を書き換えたり，特定の部分からリストを見たりする時などはFIND命令を使うことができます．

## 1.17 ファイル

ファイルとは意味を持つ情報の集まりで，例えば次のようなものがあります．

- BASIC で書かれたプログラム（プログラムファイル）
- 計算に必要な一連のデータ（データファイル）
- 日本語や英語で書かれた文章（テキストファイル）
- 機械語プログラムやメモリ内容（メモリ内容ファイル）

$N_{82}$-BASIC は外部機器との入出力や RAM 上で LOAD，SAVE などを行う時，このファイルを一つの単位として管理します．

ファイルを単位として入出力を行う場合，そのファイルにファイル番号（バッファの番号)を割り当てて入出力を行います．(ロード，セーブに関する命令やプリンタ制御命令では専用のバッファを用意してあるため，ファイル番号は一切意識しないで済みます．)

## 1.18 ファイル番号とファイルディスクリプタ

ファイルを単位として入出力を行う時，どのファイルをどの機器との間で扱うかを OPEN 文のファイルディスクリプタで指定し，入出力用のバッファをファイル番号で指定します．バッファとは入出力に必要な RAM 上のメモリ領域のことで，データはこの領域を経由して出入りします．

### 1.18.1 ファイル番号とバッファ

ファイルの入出力に必要なバッファは最高15個まで，MAXFILES 文によって確保することができます．確保されたバッファの数以内なら，同時にいくつでもファイルをオープンして入出力を行うことができます．この時，それぞれのファイルにどのバッファを割り当てるかを決めるのがファイル番号です．

### 1.18.2 ファイルディスクリプタの構成

ファイルディスクリプタは，各ファイルを入出力機器，ファイル名の違いによって区別するための名称で，次のような構成になっている一連の文字列で表されます．

"〔<デバイス名>：〕〔<オプション>〕〔<ファイル名>〕"

ファイルディスクリプタは必ず引用符（"）で囲まなければなりません．

### 1.18.3 〈デバイス名〉

〈デバイス名〉は，入出力器機を指定するためのもので，ほとんどがアルファベット3文字と区切り記号〝：〟から成っています．これを省略すると，デフォルトとして〝RAM：〟が指定されます．(CLOAD，CSAVEでは，自動的に〝CAS：〟が指定されるのでデバイス名は指定しません．)

| デバイス名 | 機器の名称 | 入力 | 出力 |
|---|---|---|---|
| LCD： | 液晶ディスプレイ | × | ○ |
| CRT： | モニタディスプレイ | × | ○ |
| CAS： | オーディオカセット | ○ | ○ |
| LPT： | プリンタ | × | ○ |
| COM： | RS-232C | ○ | ○ |
| WAND： | バーコードリーダ | ○ | × |
| 0： | 外部記憶装置 | ○ | ○ |
| 〈番号〉： | フロッピィディスク | ○ | ○ |

本体標準装備ではLCDと本体RAM以外は使えません．

### 1.18.4 〈オプション〉

〈オプション〉は通信回線(RS-232C)を指定する時のみ使用するもので，パリティチェックの有無など種々のモードを設定するのに用いられます．詳しくは第2章OPEN〝COM：〟及びユーザーズマニュアルを参照してください．

### 1.18.5 〈ファイル名〉

〈ファイル名〉は6文字以内の文字列で表されるファイルの名称と，2文字のアルファベットから成る拡張子で構成され，両者の区切りにはピリオド〝.〟が使われます．

〈ファイルの名称〉〔.拡張子〕

〈ファイルの名称〉は6文字以内の英数字，記号，カナ，グラフィック記号で構成され，6文字以内に満たない場合は残りは空白と見なされます．また7文字以上の名称はエラーになります(ただしCLOAD，CSAVE，BLOAD，BSAVEでは7文字目以降を無視します)．
その他ファイルの名称には次のような制限があります．

| | |
|---|---|
| 名称の1文字目に使える文字，記号 | 英文字，カナ，グラフィック記号及び<br>＝ ［ ］ 〈 〉 ？ ∧ ¥ |
| 名称に含まれてはならない記号 | ： . |

〈拡張子〉はファイルの形式を示すもので，〈ファイル名〉の一部と考えることができます．〈拡張子〉には次の三つがあります．

〝．ＢＡ″　内部表現に圧縮したBASICプログラムファイル．
〝．ＤＯ″　アスキー形式のBASICプログラムファイル，データファイル，テキストファイル
〝．ＣＯ″　機械語プログラムなどのメモリ内容ファイル

〈拡張子〉はこの三つだけで，その他は指定できません．〈拡張子〉の指定は省略できる場合もあり，省略した場合は〝．BA″が選択されますが，LOAD命令などで拡張子を省略した場合，もし〝．DO″ファイルしかなければそちらをロードすることになります．（BLOAD，BSAVEでは常に〝.CO″が選択されます.）

### 1.18.6　RAM上にファイルを作成する時の注意

$N_{82}$-BASICは，本体RAM上でファイルを管理するという特殊な機能を持っています．ファイルをRAM上に作成する命令は，SAVE，OPEN，BSAVEの3つがあり，これらを使って最大21個（増設RAMとバンク切り換えによってそれ以上も可能になります.）のファイルを作ることができます．

ファイルの種類による注意点は次のようになります．

〝．ＢＡ″　内部表現による保存であるため，メモリも少なくて済み，ロード，セーブにも時間がかかりません．ただしアクセス中（FILESを実行した時アスタリスク〝＊″が示される）にプログラムのエディットを行うとファイルそのものが，変更されてしまうので注意が必要になります．間違って変更されるのを防ぐには〝．DO″ファイルにしておくのが良い方法です．

〝．ＤＯ″　データファイルはこの形式でしか作れません．また〝．DO″で作られたプログラムは，ロードするのに多少時間がかかりメモリも多く使いますが，メニュー画面から直接TEXTモードに移ってエディットしない限り（つまりBASICのプログラムとして扱っている限り）は変更されません．またMERGE命令で他に融合されるプログラムはこの形式になっていなくてはなりません．

〝．ＣＯ″　BSAVE，BLOADを使わない限り入出力はできません．エディットもできません．

コマンド別に考えると

SAVE　プログラムをファイルとしてセーブします．拡張子を特に〝．DO″に指定しない限り〝．BA″ファイルを作ります．

OPEN　主に，あるプログラムの実行中に他のデータファイルをオープンしデータを読んだり，書き込んだりします．〝．DO″ファイルしか扱えません．

BSAVE　〝．CO″ファイルを作るための専用命令です．

となります．

### 1.18.7 BASICエリア

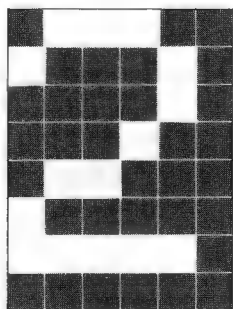
$N_{82}$-BASICの本体RAM上では，プログラムファイルの形式として〝．DO″ファイルと〝．BA″ファイルの区別があります．また，プログラムの実行時にはBASIC自身がプログラムを持つことを必要とします．このためのメモリ領域を本書では特にBASICエリアと呼んでいます．$N_{82}$-BASICでは，〝．DO″ファイル（ASCII形式）に対しては外部の機器（例えばカセットなど）に対するのとまったく同様な動作を行いますが，〝．BA″ファイル（内部表現形式）に対しては特別な動作を行います．

外部の機器に対してSAVEを行う場合には，BASICエリア上のプログラムをただ単に外部に出力しますが，RAM上に〝.BA″ファイルのSAVEを行う場合には，同じ内容のプログラムを本体RAM上に二つも作成してしまう無駄をさけるため，NEW，MENU，LOAD，CLOADなどのコマンドが実行されるまで，BASICエリア上のプログラムをRAM上に転送せず，単に名前を登録するだけしか行いません（これらのコマンド実行と同時にプログラムの内容が転送されます）．そのためRAM上に〝．BA″ファイルの形でSAVEをした後にBASIC自身が持つプログラムに編集を加えると，結果的にはSAVEされたプログラムも同様に変化してしまうのです．

RAM上の〝.BA″ファイルをLOADする場合には．BASIC身体がプログラムを読み込んだ後にRAM上のプログラムを消してしまいます．この時にも上記と同様にBASICエリア上のプログラムに編集を加えてしまうと，RAM上のプログラムも変化することになります．

このマニュアルでは，上記のようにRAM上にプログラムをSAVE，LOADしている時で，BASICエリア上には存在していてもRAM上では名前だけが残っているプログラムファイルのことを「アクセス中のファイル」と呼び特に区別をしてあります．この「アクセス」はNEW，LOAD，CLOADなどが実行されるか，電源スイッチを切るなどしてメニュー画面に戻るまで続きます．

プログラムをアクセス中である場合は，FILES命令を実行した際に，ファイル名の拡張子の後にアスタリスク〝*〞が表示されます．プログラムの改良を試みる場合には，〝.DO〞ファイルの形でSAVEを行い，原形となるファイルを常に保存するように心がけてください．

PC-8201は電源スイッチを切ってもプログラムは保存されています．BASICを起動した時点で〝アクセス中のプログラム〞が存在していた場合，いきなり新たなプログラムを入力してしまったりすると，前のプログラムは使いものにならなくなってしまいます．LISTやFILES命令を実行しBASIC自身のプログラムの状態を知る習慣をつけるようにしてください．

# コマンド・ステートメント

# 2 コマンド・ステートメント

# BEEP

| | |
|---|---|
| 機能 | 内蔵スピーカを約0.12秒間鳴らす. |
| 書式 | BEEP |
| 文例 | BEEP |
| 解説 | PRINT CHR$(7)を実行したのと同じになります. |
| 注意 | N-BASICのようにスピーカを鳴らしっぱなしにする，または止めるためのパラメータはつける事ができません. |
| 参照 | SOUND |

サンプルプログラム

```
10 REM *** BEEP ***
20 FOR I=0 TO 6
30 READ W:BEEP
40 FOR J=0 TO W:NEXT J
50 NEXT I
60 DATA 100,10,10,100,300,100,100
```

# BLOAD/BLOAD?

**機能**　機械語ファイルをロードする.

**書式**　BLOAD〔?〕"〈ファイルディスクリプタ〉"

**文例**　BLOAD "HEXCAL"

**解説**　ファイルディスクリプタのデバイス名を省略した場合はRAMが選択され,("RAM:〈ファイル名〉"と同じ)ファイル名を省略したり存在しないファイル名で命令を実行するとエラーになります. RAM上のファイルはBSAVEで作ったもので拡張子".CO"が付いていなければロードできませんが,ロードする時にはこの拡張子は省略する事ができます.
".CO"ファイルを作る時,(詳しくはBSAVEを参照)もし実行開始アドレスが指定してあれば,この".CO"ファイルはロードされた時点でサブルーチンとして実行を開始します. すなわち,このような".CO"ファイルをロードすると,あらためてEXEC文を実行する必要はなくなります. サブルーチンからは機械語命令のRETでBASICに復帰します.
デバイス名に"CAS:"を指定するとカセットテープからロードします. この場合はファイル名を省略すると最初に見つけたファイルをロードします. またカセットを対象とした場合のみBLOAD? "CAS:〈ファイル名〉"を実行する事により,カセット上のファイルとメモリの内容を比較してベリファイ(正しくセーブされたか)を行う事ができます. 正しければ"Ok",間違っていれば"Bad"と表示されます. BLOAD?は通常BSAVE実行直後に利用します. BLOAD "CAS:" 及び BLOAD? "CAS:" の実行を中断したい時は[SHIFT]+[STOP]キーを押します.
その他にオプションのデバイス名として外部記憶装置用に"0:"やフロッピィディスク用に"〈番号〉:"が用意されています.

**注意**　BLOADを実行する前に必ずCLEARの第二パラメータでその領域を確保してください. 確保していないと,?OM Errorを発生してコマンドレベルに戻ります.
BLOADをプログラム中で実行してもプログラムの実行は止まらず変数内容も影響されません.

**参照**　BSAVE, CLEAR, EXEC, 第1章ファイル, 第4章機械語プログラム

# BSAVE

| 機　能 | メモリの内容を指定したファイルにセーブする. |
|---|---|
| 書　式 | BSAVE "<ファイルディスクリプタ>",<先頭番地>,<バイト長>〔,<実行開始番地>〕 |
| 文　例 | BSAVE "BCD",61000,256 |

**解　説**

指定されたファイルに，機械語のプログラム，またはメモリの内容を書き込みます．ファイルディスクリプタのデバイス名を省略するとRAMが選択され（"RAM：<ファイル名>"と同じ），拡張子".CO"のファイルを作る事ができます(ファイル名は省略できません)．セーブする範囲は<先頭番地>，<バイト長>で指定し，文例では61000から61255番地までの内容がセーブされます．

オプションの<実行開始番地>を指定すると，".CO"ファイルそのものにこのことが書き込まれ，BLOAD文によってロードされると，すぐに機械語サブルーチンとして実行されます．

デバイスに"CAS："を指定するとカセットにセーブする事ができますが，この場合はファイル名を省略する事もできます．またBSAVE "CAS："の実行を中断したい時は[SHIFT]+[STOP]キーを押さなければなりません．

オプションとして外部記憶装置用"0："やフロッピィディスク用"<番号>："も用意されています．

BSAVEは実行を終わると，いつでもコマンドレベルに戻ります．

**参　照**

BLOAD, SAVE, 第1章ファイル, 第4章機械語プログラム及びキャラクタ定義

# CLEAR

| 機　能 | 変数の初期化およびメモリ領域の設定をする. |
|---|---|
| 書　式 | CLEAR〔<ストリング領域の大きさ>〕〔,<BASICで使うメモリの上限>〕 |
| 文　例 | CLEAR 300,60000 |

解　説

すべての数値変数を0に，文字型変数を “” (ヌルストリング) に初期化します．パラメータの指定を省略した場合は以前の値を保持しますが，初期設定はそれぞれ256，62336となっています．文字型変数の内容となるストリングの長さの合計が大きい時は，この第一パラメータを必要なだけ指定する必要があります．上限はその時に空いているメモリの量に左右されます．

第二パラメータはBASICで使うメモリの上限を設定し，機械プログラムなどで使用するメモリ空間を確保します．文例では59999番地が上限となり，60000～62335番地に機械語プログラムなどを置くことができます．下限は空いているメモリの量に左右され，また62337以上はシステムで使用するので指定できません．

CLEAR文を実行するとPASTEバッファ内のデータもクリア（消去）されます．PASTEバッファについてはユーザーズマニュアル第2章のTEXTを参照してください．

注　意

両パラメータを省略した場合，単に変数の初期化を行い，メモリ領域の設定は変わりません．第二パラメータだけを省略した時はストリング領域だけが変化しますが，第一パラメータだけを省略する事はできません．メモリ領域の設定は，新たにCLEAR文が実行されるまで変わりませんので，大きなストリング領域を取ったまま忘れていると，別のプログラムを走らせていて突然 “?OM Error” (Out of memory) エラーを起こすことがあります．

参　照

BLOAD，EXEC，DIM，第1章変数（文字型変数）

サンプルプログラム

```
10 REM ** CLEAR **
20 A$="ATW":B=486:C=7111
30 PRINT "A$=";A$;"  B=";B;"C=";C
40 PRINT "CLEAR !":BEEP
50 CLEAR
60 PRINT "A$=";A$;"  B=";B;"C=";C
```

# CLOAD/CLOAD?

**機能**　**カセットテープからプログラムファイルをロードする.**

**書式**　CLOAD〔?〕"〈ファイル名〉"

**文例**　CLOAD "DEMO"

**解説**　カセットにセーブしてあるプログラムファイルを〈ファイル名〉で探してきてメモリ内にロードします. 別のファイルを見つけた時は Skip：〈ファイル名〉のメッセージを出力して読み飛ばし, 見つけると Found：〈ファイル名〉のメッセージを出力してロードを始め, 終了すると "Ok" を出力します.
ファイル名を省略すると最初に見つけたプログラムファイルをロードします. リモート端子を差し込んであれば自動的にカセットレコーダの ON, OFF を行います.
オプションの疑問符をつけるとカセットテープ中のプログラムファイルと, BASIC エリア上のプログラムが同一であるか調べることによって, ベリファイ（正しくセーブされたか）を行います. 通常 CSAVE 実行直後に利用します. 正しければ "Ok", 違っていれば "Bad" と表示されます.

**注意**　ロード中にカセットを不要意に操作すると, "Ok"のメッセージか出てもメモリ内にロードされたプログラムが正常に動作しない場合があります. また CLOAD では拡張子やファイル名の 7 文字目以降は無視されます.
CLOAD は実行と同時に NEW をも実行しますので BASIC エリア上のプログラムはファイル名を持ったファイルとして存在していない限り消滅します. CSAVE と間違えて使うことのないように注意してください.
CLOAD の実行を中断したい時は, SHIFT + STOP キーを押さなければなりません.

**参照**　CSAVE, BLOAD, BSAVE, NEW, LOAD, SAVE, 第1章ファイル.

# CLOSE

機能　ファイルを閉じる.

書式　CLOSE〔〔#〕〈ファイル番号〉〔,〔#〕〈ファイル番号〉〕…〕

文例　CLOSE

解説　〈ファイル番号〉に対応するファイルを閉じます．以後，指定された〈ファイル番号〉は，異なるファイルを開くために利用できるようになります．また閉じられたファイルは，同じあるいは異なったファイル番号によって再び開くことができます．

CLOSE では，〈ファイル番号〉を複数指定することにより，一度に複数のファイルを閉じることができます．また〈ファイル番号〉が省略された場合には，その時，開いているファイルすべてを閉じます．

クローズしたファイルに対しては，再びオープンするまで入出力を行うことはできません．

CLOSE は，ファイルが出力用にオープンされていた場合には，バッファに残っていたデータの掃き出しを行いますので，ファイルへの出力処理を正しく終了するためには，必ずクローズを行わなくてはなりません．

END，NEW は，自動的にすべてのファイルを閉じる処理を行います．

STOP 文はファイルを閉じる作業は行いませんので注意してください．

参照　OPEN，END，NEW

# CLS

**機　能**　画面を消去する.

**書　式**　CLS

**文　例**　CLS

**解　説**　画面上の文字やグラフィックスをすべて消去します．ただしSCREEN文第2パラメータのファンクションキー表示スイッチが1の時（ONの時）は，ファンクションキー内容表示だけは残されます．

**サンプルプログラム**

```
10 FOR I=0 TO 40
20 X=RND(1)*35:Y=RND(1)*7
30 XP=RND(1)*240:YP=RND(1)*64
40 PSET(XP,YP)
50 LOCATE X,Y:PRINT "GOMI";
60 NEXT
70 LOCATE 0,0:INPUT"HIT '␣' FOR CLS";C$
80 CLS
```

# COM ON/OFF/STOP

機 能　通信回線からの割り込みの許可，禁止，停止を設定する．

書 式　1) COM ON
2) COM OFF
3) COM STOP

文 例　COM ON

解 説　コミュニケーションポート（RS-232C 回線）に外部からの通信が入ったことによる割り込みを，許可（ON）するか，禁止（OFF）するか，停止（STOP）するかを設定します．

1) 割り込みを許可します．この命令後は，コミュニケーションポートに通信が入るごとに（バッファにデータがある限り）割り込みを発生し，ON COM GOSUB 文で定義されている処理ルーチンに分岐します．
2) $N_{88}$-BASIC との互換性のために存在しますが，$N_{82}$-BASIC では3) の COM STOP と同じ働きをします．
3) 割り込みを停止します．この命令実行後は，通信があってもそのことを覚えているだけで処理ルーチンへの分岐は起こりません．しかし，その後 COM ON によって割り込みが許可されると，先程の通信によって処理ルーチンに分岐します．

参 照　ON COM GOSUB

# CONT

**機能** ストップキーの入力，またはSTOP文によって停止したプログラムの実行を再開する．

**書式** CONT

**文例** CONT

**解説** CONTは通常デバックのためにSTOP文と共に用いられます．STOP文やストップキー（または CTRL ＋ C ）の入力によりプログラムの実行を停止し，ダイレクトモードで変数の値等を調べたり変更した後に，CONT文によって実行を再開します．
実行停止中にプログラムをLISTする事はできますが，プログラム内容の変更を行った場合には，CONT文による継続実行はできません．

# CSAVE

| | |
|---|---|
| 機能 | **カセットテープにプログラムファイルをセーブする.** |
| 書式 | CSAVE "<ファイル名>" |
| 文例 | CSAVE "DEMO" |

解説　現在BASICエリア上にあるプログラムに指定したファイル名（6文字以内）を付けてカセットにセーブします．拡張子や7文字目以降は付けても無視され，常に内部表現形式のファイルを作ります．
".CO" ファイルや ".DO" ファイル（ASCII形式）をカセットにセーブする場合についてはそれぞれBSAVE，SAVEを参照してください．
CSAVEはその実行を終わると，いつでもコマンドレベルに戻ります．

注意　RAM上にあるプログラムファイルは一度LOAD命令によってBASICエリア上に移してからでなければセーブできません．CSAVEの前にLISTを実行して，目的のプログラムがBASICエリア上にある事を確かめれば，勘違いを避ける事ができます．
また，作ったばかりのプログラムをCSAVEのつもりでCLOADを実行するとNEWも同時に行なわれる為，BASICエリアのプログラムはクリアされてしまいます．注意してください．
CSAVEの実行を中断したい時は，SHIFT＋STOPキーを押さなければなりません．

参照　CLOAD，SAVE，LOAD，BSAVE，BLOAD，第1章ファイル．

# DATA

**機能** READ文で読み込まれる数値，文字定数を格納する．

**書式** DATA〈定数〉〔,〈定数〉…〕

**文例** DATA　1, CBA, 1465

**解説** DATA文は非実行文で，プログラム中のどこに置いてもかまいません．

一つのDATA文には，1行（254文字）に入るだけのデータをセットすることができます．また，一つのプログラム内には，任意の数のデータ文を置くことができます．

READ文は行番号の小さい方から順番に，DATA文中のデータを読み込んでいきます．

〈定数〉は，任意の型の定数，即ち，整数，単精度実数，倍精度実数，文字定数のいずれでもかまいません．ただし，定数式（2＊3など）は許されません．またREAD文で読み込まれる場合，READ文で指定されている変数の型は，対応するDATA文の定数の型と一致しなければなりません．

DATA文中のデータはカンマ〝,〟によって区切られますが，文字定数で，その文字列の先頭，または最後がカンマやピリオド〝.〟，意味のある空白である場合には，その文字定数の前後を引用符（″）でくくる必要があります．

RESTORE文を使えばREAD文で読み込むDATA文を行単位で指定する事ができます．

**参照** READ，RESTORE

**サンプルプログラム**

```
10 CLEAR 256:DIM A$(5),A(5):CLS
20 FOR I=0 TO 5
30 READ A$(I),A(I)
40 NEXT
50 FOR I=0 TO 5
60 LOCATE A(I),I:PRINT A$(I)
70 BEEP:NEXT
80 LOCATE 0,0
90 DATA This,5,is,11,how,16,to,21,use,25,DATA !,30
```

# DEFINT/SNG/DBL/STR

**機能**　変数の型宣言をする．

**書式**

```
DEF | INT |   <文字の範囲>
    | SNG |
    | DBL |
    | STR |
```

**文例**　DEFINT A, I-K

**解説**　<文字の範囲>で指定された文字で始まる変数名の変数を，DEFINT では整数型に，DEFSNG では単精度実数型，DEFDBL では倍精度実数型に，DEFSTR では文字型にそれぞれ定義します．

<文字の範囲>で指定される文字は英字1文字で，複数個指定する場合はハイフンでその範囲を示します．

ただし，これらによって行われた型宣言より，型宣言文字による指定の方が優先されます．また，型宣言が行われていない文字で始まる変数は，すべて単精度変数とみなされます．

**サンプルプログラム**

```
10 REM *** DEFINT/SNG/DBL/STR ***
20 DEFINT A-J,L:DEFSNG N-T
30 DEFDBL U-W:DEFSTR S,X-Z
40 A=53.9314558#:T=53.9314558#
50 W=53.9314558#:SE=" END"
60 PRINT A,T,W,SE
```

# DIM

**機能** 配列変数の要素の大きさを指定し，メモリ領域に割り当てる．

**書式** DIM 〈変数名〉(〈添字の最大値〉〔,〈添字の最大値〉…〕)

**文例** DIM A(12,2)

**解説** 配列変数の添字の最大値を設定し，同時にメモリ上にその配列の領域を割り当てます．DIM 文で宣言せずに配列変数を用いた場合，その添字の最大値は10とみなされます．もし，設定された最大値より大きい値の添字が用いられた場合は，〝? BS Error〟(Subscript out of range エラー) が起こります．

また同じ配列を2度宣言すると〝? DD Error〟(Duplicate Definition エラー) が起こります．その前に CLEAR 文を実行しておけば問題ありません．

また，DIM 文が実行された時点で，その配列のすべての要素の値は0(文字型の場合はヌルストリング) に設定されます．

**参照** 第1章 配列変数

**サンプルプログラム**

```
10 REM ** DIM **
20 PRINT"20 ｶｲ RND(1) ｦ ﾀﾞｼﾃ ｿｰﾄ ｼﾏｽ。
30 DIM R(19)
40 FOR I=0 TO 19:R(I)=RND(1):NEXT I
50 FOR I=0 TO 18:L=R(I):N=I
60 FOR J=I+1 TO 19
70 IF R(J)<L THEN L=R(J):N=J
80 NEXT J:T=R(I):R(I)=L:R(N)=T
90 NEXT I
100 FOR I=0 TO 19
110 PRINT USING" #.####### ";R(I);
120 NEXT I
```

# DSKO$（ディスク）

**機 能**　**フロッピィディスクに対しての直接書き込み．**

**書 式**　**DSKO$〈ドライブ番号〉，〈トラック番号〉，〈セクタ番号〉，〈スイッチ〉〈文字式〉**

**文 例**　**DSKO$ 2，10，1，1**

**解 説**　通常のファイル操作とは無関係に，フロッピィディスク上の指定したセクタに直接書き込みを行います．この文は，既存のディスクファイルを壊す恐れがありますので，フロッピィディスク及びファイルの構成を正しく理解した上で用いてください．これらについては，フロッピィディスク装置付属のマニュアルで説明されています．

DSKO$は，そのパラメータにより指定されたセクタの前半または後半（〈スイッチ〉で指定します）に，〈文字式〉の内容を書き込みます．〈文字式〉の長さは128文字までで，それを越えた分は無視されます．

〈スイッチ〉は0か1で指定し，0であるとセクタ前半，1の場合はセクタ後半に書き込みます．

〈トラック番号〉，〈セクタ番号〉として指定できる値の範囲は，指定された〈ドライブ番号〉のディスクドライブの種類によって異なりますが，BASICは，これらの値の範囲を自動的に調べ，不当であった場合には"? TS Error"（Bad track/sector エラー）とします．

**注 意**　フロッピィディスク装置が接続された時のみ使える命令です．

フロッピィディスク装置付属のマニュアルも必ず参照してください．

**参 照**　DSKI$，FIELD，DSKF，VARPTR

# EDIT

**機 能**　TEXT モードに移る.

**書 式**　EDIT〔〈編集開始行〉−〈編集最終行〉〕

**文 例**　EDIT 20−80

**解 説**　TEXT モードに移り，プログラムの編集を行います．行を指定しないと全行を対象とします．また〈編集開始行〉一(ハイフン)は，編集開始行から以降すべて，一(ハイフン)〈編集最終行〉は最初の行から編集最終行を対象とします．ピリオド（.）は，BASIC が現在注目している行を意味します．

**参 照**　第1章 特殊記号の使い方，第1章 TEXT モードでのエディット

# END

| | |
|---|---|
| 機　能 | **プログラムの終了を宣言する.** |
| 書　式 | END |
| 文　例 | END |

解　説　プログラムの実行を終了し，すべてのファイルをクローズしてコマンドレベルに戻ります.

END文はプログラムの実行を終了させたい所に置けばよく，また，いくつ置いてもかまいません．プログラムの最後のEND文は省略することができますが，この場合，ファイルのクローズは行われません.

参　照　STOP，CLOSE

サンプルプログラム

```
10 REM *** END ***
20 PRINT" HIT ANY KEY"
30 IF INKEY$="" THEN 30
40 CLS:LOCATE 1,3
50 FOR I=0 TO 10:READ S,L,P$
60 PRINT P$;" ";:SOUND S,L:NEXT
70 END
80 PRINT" ｺﾉ ﾌﾞﾝ ﾊ ｼﾞｯｺｳ ｻﾚﾏｾﾝ｡"
90 DATA 11172,16,THIS,11172,32,IS,11172,16,THE,11172
100 DATA 64,END,█,32,.,9394,32,MY,9952,32,ONLY,12538
110 DATA 32,FRIEND,11172,48,.,9394,16,.,11172,64,.
```

# ERROR

**機能** エラー発生のシミュレーションを行う.

**書式** ERROR <整数表記>

**文例** ERROR 200

**解説** <整数表記>の値は0から255までの範囲の数を指定します. この値が, すでにBASICでエラーコードとして使われている場合, ERROR文は, そのエラーをシミュレートし, 対応するエラーメッセージがプリントされます.
また, 未定義のエラーコードのエラーをプログラム中で条件に応じてERROR文で発生させ, ON ERROR GOTO文によって指定されたエラー処理ルーチン内で, ユーザーが定義したエラーコードとして使用することができます.

**参照** ON ERROR GOTO, 第3章ERL/ERR, 第7章 エラーコード表

**サンプルプログラム**

```
10 REM *** ERROR ***
20 ON ERROR GOTO 500
30 A=1/0
40 GOTO 0
50 NEXT
60 PRINT SQR(-2)
70 ERROR 255
80 END
500 PRINT ERL;"ｷﾞｮｳﾃﾞ"
510 IF ERR=11 THEN PRINT "Division by zero ";
520 IF ERR=8  THEN PRINT"Undefined Line number ";
530 IF ERR=1 THEN PRINT"Next without for ";
540 IF ERR=5 THEN PRINT"Illegal function call ";
550 IF ERR=255 THEN PRINT"ﾐﾃｲｷﾞ ﾉ ";
560 PRINT"ｴﾗｰ ｶﾞ ﾊｯｾｲ ｼﾏｼﾀ":PRINT
570 RESUME NEXT
```

# EXEC

| | |
|---|---|
| 機能 | 機械語サブルーチンを実行する. |
| 書式 | EXEC <開始番地> |
| 文例 | EXEC 61000 |

解説　メモリ中に用意された機械語サブルーチンに制御を移します. サブルーチンを用意する方法については第4章 機械語プログラム及び BLOAD, POKE を参照してください. <開始番地>は33468から65535の整数で指定します. 負の数を指定すると65536からその値を引いたもの（－1なら65536－1で65535となる）を指定したとみなします.

次の番地に値を POKE しておけば A, L, H 各レジスタに値を渡す事ができます. サブルーチンから BASIC に復帰後, 同じ番地を PEEK 関数で調べればその結果を受けとる事も可能です.

A　レジスタ　63911番地
L　レジスタ　63912番地
H　レジスタ　63913番地

サブルーチンからは機械語の RET 命令で BASIC に戻ります.

注意　開始番地の指定を誤ると暴走の原因になることがあります.

参照　BLOAD, 第3章 PEEK, POKE, 第4章 機械語プログラム

－18711
－3498
－17192
－52130
－32603
－31910
－31990

# FILES

**機能** RAM 上のファイルをすべて表示する

**書式** FILES〔〈デバイス番号〉〕

**文例** FILES

**解説** 〈デバイス番号〉を省略した場合 RAM 上に作ってあるすべてのファイルを，拡張子も含めたファイル名で表示します．拡張子〝.BA〟はベーシックプログラムファイル，〝.DO〟はテキストファイル，〝.CO〟はメモリ内容そのもの（例えば機械語プログラム）のファイルを意味しています．拡張子〝.BA〟の後にアスタリスクが表示されている場合は，現在アクセス中である事を意味しています．

〈デバイス番号〉に 0 を指定した場合は外部記憶装置内のファイルを表示し，1～4を指定した場合はそれをドライブ番号とするフロッピィディスク内のファイルを表示します．

**参照** 第1章ファイル

**サンプルプログラム**

```
10 REM ** FILES **
20 ON ERROR GOTO 160
30 PRINT" ﾂｷﾞﾉ ﾅｶﾃﾞLOAD,OPEN,BLOAD ｼﾀｲ ﾓﾉ ﾊ"
40 FILES
50 INPUT" ﾄﾞﾚﾃﾞｽｶ(ｶｸﾁｮｳｼﾓ ｲﾚﾃ)";N$
60 K$=RIGHT$(N$,3)
70 IF K$=".BA" THEN 110
80 IF K$=".DO" THEN 120
90 IF K$=".CO" THEN 130
100 PRINT "ﾌｧｲﾙ ﾒｲ ｼﾃｲﾁｶﾞｲ !":BEEP:GOTO 30
110 LOAD N$
120 OPEN N$ FOR INPUT AS #1:GOTO 140
130 BLOAD N$
140 INPUT"ｱﾄ ﾊ ﾄﾞｳｼﾏｼｮｳ";DM$
150 END
160 RESUME 100
```

# FOR･･･TO･･･STEP～NEXT

**機　能**　FOR 文から NEXT 文までの一連の命令を指定回数だけ繰り返して実行する．

**書　式**

```
FOR 〈変数〉=〈初期値〉 TO 〈終値〉〔STEP〈増分〉〕
   〜
NEXT 〔〔〈変数〉〕〔, 〈変数〉〕〕
```

**文　例**

```
FOR J= 0 TO 100 STEP 2
   〜
NEXT J
```

**解　説**　〈変数〉はカウンタとして使われ，最初に初期値を設定されます．そして，FOR 文以降から NEXT 文までが実行され，カウンタの値は STEP によって指定された〈増分〉だけ増減されます．次にカウンタの値が〈終値〉と比べられ，カウンタの値が〈終値〉に達していなければ，FOR 文の次の文へ戻り，同じ処理が繰り返されます．

STEP 文が省略された場合，〈増分〉は＋1とみなされます．また，〈増分〉は，負の値をとることもできます．

次の場合には，〈変数〉に〈初期値〉が代入され，1回だけ NEXT 文までの一連の命令が実行されます．

1)　〈増分〉が正の値で，〈初期値〉が〈終値〉より大きい場合．

2)　〈増分〉が負の値で，〈初期値〉が〈終値〉より小さい場合．

3)　〈初期値〉と〈終値〉が同じ値の場合．

FOR～NEXT は入れ子構造にすることができます．これは一つの FOR～NEXT のなかに更に FOR～NEXT を置くことができるということで，この場合，それぞれの〈変数〉は別のものを使わなければなりません．また，一つの FOR～NEXT は完全に他の FOR～NEXT の内部になければなりません．

**サンプルプログラム**

```
10 REM ** FOR NEXT **
20 FOR I=1 TO 5
30 FOR J=16000 TO 1000 STEP -1000
40 SOUND J,I
50 NEXT J,I
```

# FORMAT（ディスク）

**機 能** **フロッピィディスク媒体を初期化（フォーマット）する.**

**書 式** FORMAT <ドライブ番号>

**文 例** FORMAT

**解 説** フォーマットされていない新品のフロッピィディスク媒体，または異なったシステムでフォーマットされたフロッピィディスク媒体，または何らかの理由でデータを破壊してしまい，読み取り不能になったフロッピィディスク媒体を再度使用するには，この FORMAT 文を実行してフロッピィディスク媒体をフォーマットしてください.
さらに，BASIC プログラムで利用できるようにするために format ユーティリティ（ディスクのタイプによりプログラム名が異なる）で，FAT や ID などを作ることが必要です.

**注 意** フロッピィディスク装置が接続されている時のみ使える命令です.
フロッピィディスク装置付属のマニュアルも必ず参照してください.

# GOSUB～RETURN

機能　サブルーチンをコールする.

書式　GOSUB <行番号>

書式　GOSUB 1000

解説　<行番号>から始まるサブルーチンプログラムをGOSUB文によってコールし,サブルーチンプログラム内のRETURN文によって,GOSUB文の次の文へリターンします.

サブルーチンプログラムとは，他から独立したプログラムの一部分で,RETURN文で終っているものをいいます.これらは,GOSUB文によって，いつでも何度でもコールすることができます.

一つのサブルーチンの中から他のサブルーチンをコールすることもでき，これをサブルーチンの多重化と呼びます．この多重化はメモリのスタック領域の容量が許す限り行う事ができ，スタック領域が足りなくなった場合には，〝?OM Error〟(Out of memory エラー）が起こります.

サブルーチンは必ずGOSUB文によってコールされなければなりません.もし,単独で実行した場合,RETURN文に出会うと,〝?RG Error〟(RETURN without GOSUB エラー）が起こります.

また，一つのサブルーチン内に複数のRETURN文があってもかまいませんが，正しくGOSUB文と対応していなければなりません.

なお,RETURN文では<行番号>を指定して,条件に応じて特定の行へ強制的にリターンさせることができます.

参照　RETURN

サンプルプログラム

```
10 REM ** GOSUB **
20 GOSUB 50:GOSUB 80:GOSUB 110
30 END
40 REM WORK 1
50 FOR I=0 TO 9:PRINT"WORK 1":NEXT
60 BEEP:RETURN
70 REM WORK 2
80 FOR I=0 TO 9:PRINT"WORK 2":NEXT
90 BEEP:RETURN
100 REM WORK 3
110 FOR I=0 TO 9:PRINT"WORK 3":NEXT
120 BEEP:RETURN
```

# GOTO

**機　能**　指定した行へジャンプする．

**書　式**　1)　GOTO 〈行番号〉

2)　GO TO 〈行番号〉

**文　例**　GOTO 500

**解　説**　〈行番号〉で指定した行へジャンプします．

1)，2)はまったく同機能です．

**注　意**　“GO”と“TO”の間のスペースは1個のみが許されます．2個以上のスペースを入れた場合，BASICはそれをGOTO文とは解釈しません．

**参　照**　IF，GOSUB

**サンプルプログラム**

```
10 REM *** GOTO ***
20 GOTO 60
30 PRINT" SPAGHETTI.":GOTO 70
40 PRINT" CALLED A";:GOTO 30
50 PRINT" NOT MAKE";:GOTO 90
60 PRINT" THIS IS";:GOTO 40
70 PRINT:PRINT" DO";:GOTO 50
80 PRINT" PROGRAM.":GOTO 100
90 PRINT" THIS KIND OF A";:GOTO 80
100 END
```

# IF…THEN…ELSE/IF…GOTO…ELSE

**機　能**　論理式の条件判断を行う.

**書　式**

| | | | | |
|---|---|---|---|---|
| IF 〈論理式〉 | THEN | 〈文〉<br>〈行番号〉 | 〔ELSE | 〈文〉<br>〈行番号〉〕 |
| | GOTO | 〈行番号〉 | | |

**文　例**　IF A$= "y" THEN BEEP ELSE 120

**解　説**　論理式の条件によってプログラムの実行を制御します. すなわち, 〈論理式〉が真 (0以外) ならば, THEN あるいは GOTO 文以降を実行し, 偽 (0) ならば ELSE 文以降が実行されます. もし, ELSE 文以降が省略されている場合には, 次の行が実行されます.

IF…THEN…ELSE 文において, THEN 文は ELSE に続けて別の IF 文を置いて多重にすることができます. 多重にできるのは1行に書ける範囲に限られます.

**参　照**　第1章 関係演算, 論理演算

**サンプルプログラム**

```
10 REM *** IF YOU WIN ***
20 M=10000:CLS
30 PRINT " YOUR MONEY=";M
40 PRINT M;"ﾉｳﾁ ｲｸﾗ ｶｹﾙ";:INPUT K
50 K=INT(K):PRINT
60 REM ** 80 ｷﾞｮｳ ﾉ ﾖｳﾅ IF ﾉ ﾈｽﾃｨﾝｸﾞ ﾊ
70 REM ** ｱﾏﾘ ｺﾉﾏｼｲｺﾄ ﾃﾞﾊ ｱﾘﾏｾﾝ｡
80 IF K>M THEN PRINT" ｿﾝﾅﾆ ﾅｲﾖ !":BEEP:GOTO 40 ELSE IF K<0 T
HEN PRINT" ﾊﾝｿｸ !":BEEP:GOTO 40 ELSE IF K>M/2 THEN PRINT" ﾌﾄ
ｯﾊﾟﾗ !" ELSE IF K<M/100 THEN PRINT" ｹﾁ !"
90 INPUT" ｻｲﾉ ﾒ ｦ ｱﾃﾖ(1-6)";N
100 N=INT(N):PRINT
110 IF N<1 OR N>6 THEN PRINT" ﾀﾞﾒ !":BEEP:GOTO 90
120 SOUND 3000,20:R=INT(RND(1)*6)+1
130 PRINT:PRINT" ｻｲﾉﾒ ﾊ";R;"ﾃﾞｽ":PRINT
140 IF N=R THEN SOUND 4000,10:M=M+K*6:PRINT"ｱﾀﾘ !" ELSE PRIN
T" ﾊｽﾞﾚ ":SOUND 16000,10:M=M-K
150 IF M<1 THEN PRINT" ﾊｻﾝ ｼﾏｼﾀ" ELSE IF M>1E+06 THEN PRINT"
 YOU ARE MILLIONNAIRE !" ELSE 40
```

# INPUT

**機能**　指定した変数へ入力する.

**書式**　INPUT〔<"プロンプト文";>〕<変数>〔,<変数>…〕

**文例**　INPUT "Name, Add" ; N$, A$

**解説**　INPUT 文が実行されると，プログラムはキーボードからの入力待ちの状態となります．プロンプト文を付ける時は引用符で囲み，セミコロンの後に変数名を続けます．プロンプト文の内容として引用符（"）は使えません．変数はカンマで区切ればいくつでも続ける事ができますが，プロンプト文と合わせてプログラムの1行以内に収まらなければなりません．複数個の変数を一度に入力する場合は，必要な数だけカンマで区切ってやれば入力できますが，一つ一つリターンキーを押しながら入力してゆくと，2回目からは"??"が表示され，残りの変数の入力を要求してきます．またデータが<変数>の個数より多い多合には"? Extra ignored" のメッセージを表示して次の文へ制御が移ります.

数値変数に文字列を入力しようとすると"? Redo from start" のメッセージを表示し再度入力を要求してきます．また，リターンキーだけを押した場合は，何も行われなかったと見なされ，<変数>の内容は INPUT 文を実行する直前と変わりません．複数個のデータ入力の途中でリターンキーだけを押した場合は，残りのすべての<変数>に対する入力がスキップされ，次の行へ制御が移ります.

文字型変数に文字列を代入する場合，カンマや文字列の前後に意味のある空白を入力したい時は，引用符（"）で文字列を囲む必要があります．この場合は文字として（"）を入力することはできません．これ以外の場合には，文字列を引用符で囲む必要はありません.

文字列の先頭に引用符をつけなければ，それ以後つけた（"）はすべて文字列の一部とすることができます.

**参照**　LINE INPUT，INPUT #

**サンプルプログラム**

```
10 REM ** INPUT **
20 INPUT"A,B,C";A,B,C
30 PRINT "A=";D;"B=";E;"C=";F
35 PRINT "Ok"
36 A$=INPUT$(1)
40 'FOR I=0 TO 900:NEXT
50 BEEP:PRINT "ｼﾞｬ ﾅｶｯﾀ !"
60 PRINT "A=";A;"B=";B;"C=";C
```

# INPUT♯

| | |
|---|---|
| 機　能 | ファイルからデータを読み込む. |
| 書　式 | INPUT #<ファイル番号>, <変数>〔, <変数>…〕 |
| 文　例 | INPUT #1, A, B, C$ |

解　説

指定したファイル (RAM, カセットテープなど) からデータを入力することと, 疑問符〝?〟が出力されないことを除けば, INPUT 文とはほぼ同じです.

<ファイル番号>は, OPEN 文によって入力モードとしてオープンされたファイル番号でなければなりません. またファイルに書き込まれているデータは, INPUT 文に対して必要なデータと同様に, 正しい形になっていなければなりません.

INPUT #文によってファイルからデータを読み込む場合には, データはファイルに用意されていなければなりません. もし, データを読み尽くした後に INPUT #文を実行すると〝?EF Error〟が発生します. これを防ぐには EOF 関数を使います. この点に気をつければ, 複数の PRINT #文で書き込んだデータを, 一つの INPUT #文で読み込むことも可能です.

参　照　PRINT #, INPUT, LINEINPUT #, OPEN, 第3章 EOF

# KEY

機　能　**キーボードの上部にあるプログラマブルファンクションキーの内容を定義する.**

書　式　KEY <キー番号>, <文字列>

文　例　KEY 1, "LOAD"

解　説　ファンクションキーは五つあり, シフトモードを加えると合計10の<文字列>を記憶させておくことができます. 各ファンクションキーは最大15文字までの文字列およびコントロール文字を定義でき, キーボードから入力できない文字は, CHR$ 関数をプラス記号で継いで使います. <キー番号>は1から10までで, 6から10まではシフトモードです.

注　意　一度定義したファンクションキーの内容は, あらたに定義し直すか, コールドスタートされるまで変わりません.

参　照　第7章 キャラクタコード表

サンプルプログラム

```
10 REM ** KEY **
20 A$(0)=""
30 A$(1)="Load "+CHR$(34)
40 A$(2)="Save "+CHR$(34)
50 A$(3)="Files"+CHR$(13)
60 A$(4)="List "
70 A$(5)="Run  "+CHR$(13)
80 FOR I=1 TO 59
90 KEY (I MOD 5)+1,A$(I MOD 6)
100 NEXT
```

# KILL

| 機　能 | 指定したファイルを削除する. |
|---|---|
| 書　式 | KILL "<ファイルディスクリプタ>" |
| 文　例 | KILL "SAMPLE .BA" |

解　説　<ファイルディスクリプタ>で指定したファイルを削除します.<ファイルディスクリプタ>のデバイス名を省略すると, "RAM:" が選択され, ファイル名は拡張子(BA, DO, CO)も必ず指定しなければなりません. また現在アクセス中の（FILESを実行した時アスタリスク "*" が示される）".BA" ファイルやOPENしたままの ".DO" ファイルに対してKILLを実行する事はできません. KILLを実行するとBASICは, いつでもコマンドレベルに戻ります.
<ファイルディスクリプタ>のデバイス名は"RAM:"の他は, オプションとしての "<番号>:"(フロッピィディスク用)と "0:"(外部記憶装置用) しか指定できません.

参　照　LOAD, SAVE, 第1章 ファイル

# LET

**機　能**　変数に値を代入する．

**書　式**　〔LET〕<変数>＝<式>

**文　例**　LET　A＝10

**解　説**　キーワード LET はなくてもかまいません．つまり<式>の値を<変数>に代入するには，等号だけでもよいのです．
<式>の内容は，数値，文字のいかなる型であってもかまいません．ただし，文字型から数値型への代入またはその逆は許されません．型の異なる数値型変数への代入では左辺の型に変換されます．

**サンプルプログラム**

```
10 REM ** LET **
20 BE=26:IT=810
30 LET IT=BE
40 PRINT IT,BE
```

# LINE (ディスク) (CRT)

**機　能**　ドットを使って線や箱を書く.

**書　式**　LINE〔(〈水平座標1〉,〈垂直座標1〉)〕−(〈水平座標2〉,〈垂直座標2〉)〔,〔スイッチ〉〕, B〔F〕〕

**文　例**　LINE (0, 0)−(239, 63), 1, BF

**解　説**　画面上にドットを使って，指定した2点間を結ぶ線や箱を描きます．スイッチは偶数（0〜254）がOFF，奇数（1〜255）がONで，省略するとONが選択されます．〝B″オプションをつけると，指定した2点を対角線に持つ箱を描き，更に〝F″オプションをつけるとその中をドットで塗りつぶすことができます．スイッチを指定する値は0以上，255以下で，それ以外ではエラーになります．また(〈水平座標1〉,〈垂直座標1〉)は省略すると，最後に参照されたグラフィック座標（直前に実行したLINE文の〈水平座標2〉，〈垂直座標2〉など）が代入されます．

**注　意**　LINEはCRTまたはディスクを接続していなければ使えません．

# LINE INPUT

**機能**　1行を単位とした文字列を入力する.

**書式**　LINE INPUT〔"<プロンプト文>";〕<文字型変数>

**文例**　LINE INPUT "WHAT?"; A$

**解説**　キーボードから255文字以内の文字列を,1行全体を区切ることなく<文字型変数>に代入します.<プロンプト文>は入力を受ける前に表示される文章です.疑問符"?"は表示されません.プロンプト文以降[↵]キーが押されるまでに押されたすべてのキー入力が<文字型変数>に代入されます.したがって通常のINPUT文では入力できない区切り記号(カンマ,引用符)を入力することができます.

LINE INPUT文の実行は[STOP]キーの入力によって中断することができます.この場合はBASICはコマンド入力待ちの状態になりますが,CONTコマンドの入力によってLINE INPUTの実行をやりなおすことができます.

**参照**　INPUT

**サンプルプログラム**

```
10 REM ** LINEINPUT **
20 PRINT "LINEINPUT ﾊ ";CHR$(44);" ﾔ ";CHR$(34);" ﾅﾄﾞﾓ ｲﾚﾗﾚﾏ
ｽ｡"
30 LINEINPUT A$
40 FOR I=1 TO LEN(A$)
50 PRINT MID$(A$,I,1);
60 FOR T=0 TO 200:NEXT T
70 NEXT I
```

# LINE INPUT♯

| | |
|---|---|
| 機　能 | ファイルから1行を単位とした文字列を入力する. |
| 書　式 | LINE INPUT #<ファイル番号>, <文字型変数> |
| 文　例 | LINE INPUT #1, A$ |

**解　説**

LINE INPUT # 文は，ファイルから，キャリッジリターンまでの1行全体の文字列（255文字以内）を区切ることなく<文字型変数>に代入します.
<ファイル番号>は，OPEN 文によって入力モードとしてオープンされたファイルの番号でなければなりません.
LINE INPUT # 文は，アスキー形式でセーブしたプログラムを他のプログラムでデータとして読み込むときなどに有効です.

**参　照**

INPUT #

# LIST/LLIST

機 能　**メモリ内にあるプログラムの全部または一部をリストアウトする.**

書 式　1) LIST〔<行番号>〕〔-<行番号>〕

2) LLIST〔<行番号>〕〔-<行番号>〕

文 例　LIST 100-200

LIST

解 説　プログラムの内容を，1)はディスプレイに，2)はプリンタにリストアウトします. LIST コマンドは実行し終わると，いつでもコマンドレベルに戻ります.

<行番号>が省略された場合は，最も若い行番号のプログラムによりリストが始まります（リストはプログラムの終り，またはストップキーの入力により終ります). 最初の行番号のみが指定された場合には，その指定された行だけがリストされます. 最初の行番号とそれに続くハイフンまでが指定された場合は，その行番号に始まり，それよりも大きい行番号のすべてがリストされます. ハイフンとそれに続く2番目の行番号だけが指定された場合には，プログラムの始めからその行番号までの行がリストされます. ピリオド（.）は BASIC が現在注目している行を意味します. LLIST は出力先がプリンタであることを除いて LIST と同じです. また SAVE "LPT:"は結果としてファイル名を出力後，LLIST を実行したのと同じことになります.

注 意　LLIST 命令は外部機器に対する出力なので，実行を中止するためには SHIFT + STOP キーを押さなければなりません.

参 照　第1章 特殊記号の使い方

# LOAD

機 能　**プログラムファイルを BASIC エリアにロードする.**

書 式　LOAD "<ファイルディスクリプタ>"〔, R〕

文 例　LOAD "CAS : EXPLOD. BA", R

解 説　ファイルディスクリプタのデバイス名を省略した場合は，RAM が選択され ("RAM : " と同じ)，ファイル名を省略したり，存在しないファイル名で命令を実行するとエラーになります．RAM のファイルは SAVE 命令で作ったもので，拡張子 ".BA" か ".DO" の付いているものしかロードできませんが，ロードする時にはこの拡張子は省略することができます．この時もし同じファイル名で拡張子だけが ".BA", ".DO" として異なっている二つのファイルがあると，".BA"が優先的にロードされます．ただし TEXT モードで作った DO ファイルで，BASIC プログラム以外のファイルやデータファイルはロードできません．ロード終了後はコマンドレベルになりますが，オプションの ", R" をつけるとロード終了と同時にプログラムを実行します．
デバイス名に "CAS : " を指定するとカセットテープからロードします．この場合はファイル名を省略すると最初に見つけたプログラムファイルをロードします．LOAD "CAS : " の実行を中断したい時は，SHIFT + STOP キーを押さなければなりません．
デバイス名に "COM : " を指定した場合は対象が RS-232C 回線となり，ファイル名を指定することはできませんが，通信形式の指定をすることができます．これについては OPEN "COM : " を参照してください．
その他にオプションとして外部記憶装置用に "0 : "，フロッピィディスク用に "<番号> : " が用意されています．

注 意　LOAD は実行に先だって NEW を実行するので，現在の変数やプログラムはすべてクリアされます．

参 照　BLOAD，CLOAD，SAVE，第1章 ファイル

# LOCATE

| | |
|---|---|
| 機能 | **カーソルの位置を指定する.** |
| 書式 | LOCATE <水平座標>,<垂直座標> |
| 文例 | LOCATE 20, 5 |

**解説** カーソルを画面上の指定した位置へ移動します. 座標の範囲は, LCDを対象とした場合は<水平座標>0～39, <垂直座標>0～7となります. 両座標共に0～255の範囲外で指定するとエラーになり, 39以上の<水平座標>はすべて39として, 7以上の<垂直座標>はすべて7(ファンクションキー表示中は6)として処理されます.
CRTの場合はWIDTH文で指定した画面の大きさに左右されます.

**注意** LOCATEはキャラクタ座標を指定するので, ドットグラフィクスとは何の関連も持ちません.

**参照** WIDTH

**サンプルプログラム**

```
10 REM ** LOCATE **
20 SCREEN 0,0:CLS
30 LOCATE 10,7:PRINT "X=";X;
40 LOCATE 20,7:PRINT "Y=";Y;
50 X=INT(RND(1)*39):Y=INT(RND(1)*7)
60 LOCATE X,Y:PRINT "ﾙﾝ";
70 FOR I=0 TO 300:NEXT
80 LOCATE X,Y:PRINT "  ";
90 GOTO 30
```

# MAXFILES

| | |
|---|---|
| 機　能 | OPEN できるファイル数を設定する. |
| 書　式 | MAXFILES=<ファイル数> |
| 文　例 | MAXFILES= 3 |

解　説　OPENできるファイル数は，コールドスタート時に 1 に設定されます．複数のファイルを同時に OPEN するためにはその上限を MAXFILES 文で指定します．<ファイル数>の範囲は 0 から15で，一度指定すると再び指定があるかコールドスタートされるまでその値を保持します．

参　照　OPEN，CLOSE，第 1 章 ファイル

# MENU

| | |
|---|---|
| 機 能 | MENU画面に戻る. |
| 書 式 | MENU |
| 文 例 | MENU |

解 説 すべての変数をクリアし，MENU画面に戻ります.
この時もしBASICエリア上にアクセス中のファイル（FILESを実行するとアスタリスク"*"が示される）があるとアクセスは終了し，BASICエリアは初期状態に戻ります．その他の場合はBASICエリア上のプログラムは保存され，再びBASICモードに入る事によって実行可能となります.

# MERGE

機能　二つのプログラムを融合する.

書式　MERGE <ファイルディスクリプタ>

文例　MERGE "CAS : PARTS. DO"

解説　現在BASICエリア上にあるプログラム（LISTできる状態にある）と，ファイルディスクリプタで指定したプログラムを一つにします．ファイルディスクリプタのデバイス名は省略するとRAMが選択され，ファイル名の省略はできません．"CAS："(カセットテープ)を指定した場合，ファイル名を省略すると最初に見つけたプログラムを融合します．

"COM："（RS-232C回線）を指定した場合，ファイル名は使えませんが，通信形式の指定ができます．これについてはOPEN "COM："を参照してください．いずれの場合も，指定されるプログラムはアスキーセーブ（DOファイル）でなければなりません．

両方のプログラムに同一の行番号があった場合，指定されたプログラムの方が優先され，BASICエリア側の行は置き換えられてしまうので必要ならRENUMしておきます．

デバイス名のオプションとして外部記憶装置用に"0："，フロッピィディスク用には"<番号>："も用意されています．MERGEは，実行を終えるとすべてのファイルを閉じ，すべての変数を初期化してコマンドレベルに戻ります．

参照　SAVE，RENUM，第1章 ファイル．

# MOTOR

**機能**　カセットテープレコーダのモータのON，OFFを制御する．

**書式**　MOTOR <スイッチ>

**文例**　MOTOR 1

**解説**　<スイッチ>を0にすればモータはOFFとなり，0以外の値にすればONとなります．256以上の値を指定するとエラーになります．

**サンプルプログラム**

```
10 REM ** MOTOR **
20 MOTOR 0
30 PRINT "TAPE RECORDER ﾆ ｽｷﾅ ｵﾝｶﾞｸ ﾃｰﾌﾟ"
40 PRINT "ｦ ｾｯﾄ ｼﾃｸﾀﾞｻｲ｡"
50 PRINT "REMOTE ﾀﾝｼ(ｸﾛ) ﾀﾞｹｦ ﾂﾅｲﾃﾞｵｸ｡"
60 PRINT "HIT ANY KEY TO START !"
70 IF INKEY$="" THEN 70
80 MOTOR 1
90 PRINT "HIT ANY KEY TO STOP !"
100 IF INKEY$="" THEN 100
110 MOTOR 0:GOTO 60
```

# NAME

**機　能**　**ファイル名をつけかえる.**

**書　式**　**NAME "〈旧ファイル名〉" AS "〈新ファイル名〉"**

**文　例**　**NAME "OLD.BA" AS "NEW.BA"**

**解　説**　RAM 上のファイル名を変更します．ファイル名の指定は新旧どちらも拡張子まで必ず指定し，RAM 上に存在しないファイル名を旧ファイル名に指定したり，逆に使われているファイル名を新ファイル名に用いるとエラーになります．また，両ファイル名の拡張子は一致していなければなりません．

オプションのフロッピィディスクや外部記憶装置が接続されている場合は，ファイル名の前にデバイス名"〈番号〉："(フロッピィディスク)，"0："(外部記憶装置)を付ける事によってそれらのファイル名をつけかえる事も可能です．

**参　照**　第1章 ファイル

# NEW

機能　BASICエリアにあるプログラムを抹消し，すべての変数をクリアする．

書式　NEW

文例　NEW

解説　NEWコマンドは，コマンドレベルにあるときに新しいプログラムを入力する前に実行します．NEWコマンドの実行が終わると，いつでもコマンドレベルに戻ります．

NEWコマンドは，OPENされているファイルがある場合は，それを自動的にすべてCLOSEします．また，アクセス中のファイル（FILESを実行するとアスタリスク〝*〟が示される）がある時は，そのアクセスが停止されます．

サンプルプログラム

```
10 REM ** NEW **
20 REM コノ プ゜ログ゛ラム ハ RUN スルト シ゛サツ シマス。
30 PRINT" プ゜ログ゛ラム コ゛ロシ"
40 BEEP:BEEP
50 NEW
```

# ON COM GOSUB

**機 能** 通信回線からの割り込みが発生した時，分岐する処理ルーチンの開始行を定義する．

**書 式** ON COM GOSUB <行番号>

**文 例** ON COM GOSUB 2000

**解 説** 通信回線から RS-232C のコミュニケーションポートに入力割り込みがあった時，分岐する処理ルーチンの開始行を定義します．処理ルーチンからの復帰は，一般のサブルーチンの場合と同じで RETURN 命令によって行います．

**参 照** COM ON／OFF／STOP，OPEN，RETURN

**サンプルプログラム**

```
10 REM *** COM: ***
20 MAXFILES=2
30 OPEN"COM:"FOR INPUT AS #1
40 OPEN"COM:"FOR OUTPUT AS #2
50 ON COM GOSUB 110
60 COM ON
70 A$=INKEY$
80 IF A$="" GOTO 70
90 PRINT A$;:PRINT#2,A$;
100 GOTO 70
110 B$=INPUT$(1,#1)
120 PRINT B$;
130 RETURN
```

# ON ERROR GOTO~RESUME

機能　エラートラップを可能にし，エラー処理ルーチンの開始行を定義する．

書式　ON ERROR GOTO <行番号>

文例　ON ERROR GOTO 1000

解説　エラートラップ機能が可能になると，エラーが検出された時に指定されたエラー処理ルーチンへプログラムの制御が移ります．もし<行番号>の行が存在しなければ，〝?UL Error〟が起こります．エラートラップ機能を禁止するには，ON ERROR GOTO 0を実行してください．その後はエラーメッセージを表示し，プログラムの実行は停止します．また，エラー処理ルーチンにON　ERROR　GOTO 0の文がある場合には，BASICはエラートラップの原因となったエラーのエラーメッセージを表示し，プログラムの実行を停止します．すべてのエラー処理ルーチンは，エラー回復処理を行わないエラーをトラップした場合にON ERROR GOTO 0を実行するようにしておくことをお勧めします．

注意　エラー処理サブルーチンが実行中にエラーが起きた場合には，それに対するエラーメッセージが表示され，実行は停止します．エラー処理サブルーチンの中ではエラートラップは起こりません．

参照　RESUME, ERROR

# ON･･･GOTO/ON･･･GOSUB

**機能**　指定されたいくつかの行に分岐する.

**書式**　ON 〈式〉 GOSUB 〈行番号〉〔,〈行番号〉…〕

ON 〈式〉 GOTO 〈行番号〉〔,〈行番号〉…〕

**文例**　ON A GOSUB 100,200,300,400

ON A GOTO 100,200,300,400

**解説**　〈式〉の値がプログラムのどの行番号の行に分岐するかを決定します.〈行番号〉の並びは1から始まる数に対応します.たとえば,〈式〉の値が3であったなら,行番号の並びの3番目の行が分岐先となります.

ON…GOSUB文では,並びの各行番号はサブルーチンの最初の行の行番号でなければなりません.

〈式〉の値が負になった場合には,〝?FC Error〟(Illegal function call エラー)が起こりますが,値が0または並んでいる行番号の個数より大きくなった場合には,次の行へプログラムの制御が移り,エラーは起こりません.

**サンプルプログラム**

```
10 REM ** ON GOSUB **
20 INPUT" ﾅﾝﾆﾝﾃﾞｽｶ";N
30 ON N GOSUB 50,70,90,110
40 IF N>4 THEN 120
45 GOTO 20
50 PRINT"PC-8201 ﾃﾞｱｿﾝﾃﾞﾖ"
60 BEEP:FOR I=0 TO 300:NEXT:RETURN
70 PRINT"ﾁｪｽﾅﾄﾞ ﾊ ?"
80 BEEP:FOR I=0 TO 300:NEXT:RETURN
90 PRINT"ｼﾞｬﾝｹﾝ ﾎﾟｲ !"
100 BEEP:FOR I=0 TO 300:NEXT:RETURN
110 PRINT"ﾏｰｼﾞｬﾝ ｼｮｳ !"
115 BEEP:FOR I=0 TO 300:NEXT:RETURN
120 PRINT" ﾐﾝﾅﾃﾞ ﾉﾐﾆ ｲｷﾏｽｶ･･･"
130 END
```

# OPEN

**機能** ファイルを開く

**書式** OPEN "〈ファイルディスクリプタ〉" FOR〈モード〉 AS 〔#〕〈ファイル番号〉

**文例** OPEN "SESAME" FOR OUTPUT AS #1

**解説** 〈ファイルディスクリプタ〉で指定したファイルを，指定した〈ファイル番号〉で開きます．〈ファイル番号〉は，MAXFILESで指定した範囲の1～15が使えますが，既に開かれているファイルが利用している番号を用いる事はできません．開かれたファイルの入出力は，以後〈ファイル番号〉を指定する事により行います．

〈モード〉の指定は，出力には "OUTPUT"，入力には "INPUT"，追加出力には "APPEND" とします．〈ファイルディスクリプタ〉のデバイス名は，省略すると"RAM："を意味し，ファイル名の省略はできません．デバイス名として "CAS：" を指定すると，カセットテープが対象となります．この時，ファイル名を省略した場合は，入力モードでは最初に見つけたファイルを開き，出力モードではファイル名のないファイルを新しく作ります．OPEN "CAS：" の実行を中断したい時は，[SHIFT]＋[STOP]を押さなければなりません．

出力モードではどんなファイル名を指定しても，常に新しいファイルを作る事を意味し，既にあるファイル名を指定するとOPENした時点でその内容はすべてクリアされてしまうので注意が必要です（ただしカセットではこの限りではありません）．追加出力モードでは，開かれたファイルの最後に続けて出力してゆくことができますが，カセットを対象にした場合はこのモードは実行できません．

**注意** RS-232C回線を対象としたOPEN "COM:" については別項を設けてあります．

**参照** OPEN "COM:"，CLOSE，第1章ファイル

**サンプルプログラム**

```
10 REM ** OPEN **
20 OPEN "SESAME" FOR OUTPUT AS #1
30 PRINT #1,"ﾋﾗｹ ｺﾞﾏ !"
40 PRINT #1,"ﾄｼﾞﾛ ｺﾞﾏ !"
50 CLOSE
60 OPEN "SESAME" FOR INPUT AS #1
70 INPUT #1,A$:PRINT A$:SOUND 2000,20
80 INPUT #1,A$:PRINT A$:SOUND 5000,20
90 CLOSE
100 PRINT "SESAME ﾌｧｲﾙ ｶﾞ ﾃﾞｷﾃ ｲﾏｽ｡"
110 PRINT "FILES":FILES
```

# OPEN "COM:"

**機能**　RS-232C回線を開く

**書式**　OPEN "COM:〔〈CPBSXS〉〕" FOR〈モード〉 AS〔#〕〈ファイル番号〉

**文例**　OPEN "COM: 9N82XN" FOR INPUT AS #1

**解説**　RS-232C回線の通信形式を設定しファイルとしてOPENします．〈モード〉,〈ファイル番号〉に関しては，通常のOPEN文と同様ですので，そちらを参照してください．ただし，追加出力モードは指定できません．

COM:の後に続く〈CPBSXS〉は，通信回線の形式の設定を行う6文字のパラメータで，それぞれ次のような意味を持ちます．

| パラメータ | 設定値 |
|---|---|
| C：通信速度 | 1：75bps |
| | 2：110bps |
| | 3：300bps |
| | 4：600bps |
| | 5：1200bps |
| | 6：2400bps |
| | 7：4800bps |
| | 8：9600bps |
| | 9：19200bps |
| P：パリティ | N：パリティなし |
| | E：偶数パリティ |
| | O：奇数パリティ |
| | I：パリティビット無視 |
| B：ワード長 | 6：6ビット長 |
| | 7：7ビット長 |
| | 8：8ビット長 |
| S：ストップビット | 1：1ストップビット |
| | 2：2ストップビット |
| X：Xパラメータによる制御 | X：X制御を行う |
| | N：制御を行わない |

| | |
|---|---|
| S：シフトイン／アウト・シーケンス<br>による制御 | S：制御を行う<br>N：制御を行わない |

この〈CPBSXS〉は，省略することも可能で，省略した場合には，最後に設定された値が有効となります．

BASICでRS-232C回線を使用する場合，送・受信のためには，ファイルを二つOPENしなければなりません．この時，通信形式の設定は，どちらかのOPEN文で指定してもかまいません．両方で指定した場合には，後の設定が有効になります．

Xパラメータによる制御を行う時でも，そのためにだけ出力用にファイルをOPENする必要はありません．入力用のファイルをOPENするだけでも，CTRL-S，CTRL-Qを送信することが可能です．
RS-232C回線は，カセットレコーダと同時に使用することはできません．

その他，注意点などに関しては，ユーザースマニュアルのTELCOMの解説を参照してください．

**参　照**　OPEN，COM ON／OFF／STOP，ユーザーズマニュアル
第5章TELCOM

# OUT

**機　能**　指定した出力ポートに1バイトのデータを送る.

**書　式**　OUT <ポートアドレス>,<データ>

**文　例**　OUT 1,32

**解　説**　<ポートアドレス>はポートの番号,<データ>はそのポートに出力するデータです. これらは共に0から255までの整数で指定します.

**注　意**　OUT文は十分なハードウェアの知識なしに使うと, BASICが正常に動作しなくなることがあります.

# POKE

| | |
|---|---|
| 機 能 | **メモリ上の指定番地へデータを書き込む.** |
| 書 式 | POKE <番地>, <データ> |
| 文 例 | POKE 61400,201 |

**解 説**　指定されたメモリ上の番地に1バイトのデータを書き込みます. <番地>は2バイトの整数（0～65535), <データ>は1バイトの整数（0～255)で指定します. POKE文は, その逆の働きをするPEEK関数と組み合わせて機械語サブルーチンとの数値の受け渡しをする時などに使います.

**注 意**　POKE文は十分な機械語の知識なしに使うと, プログラムやファイルを壊してしまう事があります. POKE文の実行後, コンピュータの動作が正常でなくなった場合, リセットしなければ回復できないことがあります.

**参 照**　第3章 PEEK, 第4章 機械語プログラム

# POWER

| | |
|---|---|
| 機　能 | 自動的に電源を切る. |
| 書　式 | POWER <タイマー><br>POWER OFF 〔, RESUME〕<br>POWER CONT |
| 文　例 | POWER 200<br>POWER OFF, RESUME<br>POWER CONT |

解　説

<タイマー>は10から255の値で指定し，1単位6秒にあたります．この命令の実行後，<タイマー>で指定した時間（文例では20分）キーボードから入力が行われないことがあると，自動的に電源スイッチを OFF します．コンピュータの制御が BASIC 以外に移った時（TEXT モードなど）もこの命令は有効なので，特に電源にバッテリを使っている時などに，スイッチを切り忘れてバッテリが消耗するような事態を避ける事ができます．ただし BASIC プログラムの実行が続いている時はこの限りではありません．

POWER OFF を実行した場合は即座にスイッチを OFF しますが，オプションの〝, RESUME〟をつけておくと，再びスイッチを入れることによってスイッチ OFF が行われた時の状態に，変数の内容なども含めて復帰することができます．このためプログラム中で用いれば，長時間の計算結果などが出た時にスイッチを切り，後で好きな時にその結果を見ることもできます．

POWER CONT を実行すると以後，再度の指定があるまでは自動電源スイッチ OFF の機能は失なわれます．電源にアダプタなどを用いていたり，つけっぱなしにしておく必要があれば利用します．コールドスタート時には POWER 100（10分）に設定されています．

注　意

一度定義した <タイマー> の値は，あらたに設定し直すか，コールドスタートされるまで変化しません．

# PRESET

**機能** 画面上の任意のドットをリセットする.

**書式** PRESET (<水平座標>,<垂直座標>[,<ファンクションコード>])

**文例** PRESET (80, 32)

**解説** 指定した座標のドットをリセットします.<水平座標>,<垂直座標>,<ファンクションコード> は共に0から255の範囲でないとエラーになります.
また，LCD画面のドット座標系は239×63となっており，<水平座標>は239以上はすべて239として，<垂直座標>は63以上はすべて63として処理されます.既に表示されているキャラクタをドット単位でリセットすることも可能です.
<ファンクションコード> に偶数（0～254）を指定するとPRESETの機能は反転し，PSETとまったく同じ働きをします（ドットをセットする).奇数（1～255）を指定した場合は，省略した時と同様にドットのリセットを行います.
CRTが接続されている場合は，パラメータの範囲や <ファンクションコード> の役割が異なっていますのでCRT付属のマニュアルを参照してください.

**注意** 画面がスクロールするとドットグラフィクスはすべてクリアされます.
画面の座標系は左上隅が（0，0）です.

**参照** PSET, LINE

**サンプルプログラム**

```
10 REM ** PRESET **
20 PRINT "      These sentences will"
30 PRINT
40 PRINT "       disappear slowly"
50 PRINT
60 PRINT "      by the effects of"
70 PRINT
80 PRINT "       PRESET !";
90 FOR Y=0 TO 55:FOR X=30 TO 160
100 PRESET(X,Y):NEXT X,Y
```

# PRINT/LPRINT

**機能**　画面またはプリンタに情報を出力する.

**書式**　PRINT〔<式>…〕

LPRINT〔<式>…〕

**文例**　PRINT "ABC"

**解説**　指定した式の値や文字列を"PRINT"は画面に，"LPRINT"はプリンタに出力します．<式>が省略されている場合は改行のみを行います．式の値や文字列を表示する領域は，あらかじめ各行を14文字毎に分割して定められており，区切り記号にカンマを使うと次の領域の始めから，またセミコロンを使うと直前にプリントしたもののすぐ後ろに続いて次の値や文字列が表示されます．区切りの記号として空白" "を用いた場合もセミコロンと同様の働きをします．

<式>の最後にセミコロン及びカンマを指定すると改行動作をおこさず，その行で次の PRINT 文による出力を続行します．

**注意**　変数と（"）で囲まれた文字列との区切りに限って",", ";", " "を省略することができ，このときはセミコロンと同様の働きをします．

数値を表示した場合，その後ろに必ず空白が一つ挿入されます．また，数の前に符号のための桁を確保します．単精度の数値で，指数形式でなくても 6 桁以下の桁数で精度に影響を及ぼさず表示できるものは，実数形式で表示されます．

同様に倍精度の数値は16桁以下の桁数で精度に影響を及ぼさず表示できるものは，実数形式の表示になります．

この命令のキーワード"PRINT"の簡略形として疑問符"?"を使用することが許されます．

表示する文字列の長さ，数値の長さが現在のカーソルの位置より後方にとれない（それを表示すると次の行にわたってしまう）場合，改行してそれを表示します．

**サンプルプログラム**

```
10 REM *** Print ***
20 PRINT "ｶｲｷﾞｮｳ ｦ"
30 PRINT "ｼﾀｸ";"ﾅｹﾚﾊﾞ";
40 PRINT "ｾﾐｺﾛﾝ｡"
50 PRINT "ｳｹﾃ","ｶｸﾅﾗ";
60 PRINT "ｶﾝﾏ","ﾂｶｵｳ｡",
```

# PRINT#

| | |
|---|---|
| 機能 | ファイルにデータを出力する. |
| 書式 | PRINT #<ファイル番号>, 〔<式>…〕 |
| 文例 | PRINT # 2 , A, B, C |

**解説** <ファイル番号>は,そのファイルがOPEN文によって出力モードとしてオープンされたときに指定された番号です.

<式>はファイルに書き込まれる数値または文字式です.

PRINT #文はデータの圧縮を行わず，PRINT文で画面へ出力するものとまったく同じものを出力します. これらのデータがファイルから正しく入力できるように，ファイルに書き込まれたデータは適切に区切られるよう注意してください.

<式>の並びの中の数値はセミコロンで区切るようにしてください.

例えば,

PRINT #1, A ; B ; C ; X ; Y ; Z

区切りに記号にカンマを使うとプリント領域の間に挿入される余分な空白もファイルに書き込んでしまいます.

<式>の並びの中の文字表記もセミコロンで区切ってください. ファイル上に文字表記を正しく書き込むためには，<式>の並びの中に独立した区切り記号を入れてください.

例をあげると，A$ = "CAMERA" そして，B$ = "93604 − 1" のときの次の文,

PRINT #1, A$ ; B$

はファイルにCAMERA93604 − 1と書き込みます. これは区切り記号がありませんから2つの別の文字列として入力することはできません. この問題を解決するには,次のようにPRINT #文中に独立した区切り記号を入れてください.

PRINT #1, A$ ; "," ; B$

これによってファイルに書き込まれるイメージは,

CAMERA, 93604 − 1

となり，2つの文字型変数として読み込むことができます.

もし文字列それ自身がデータとしてカンマ，セミコロン，意味のあるはじめの空白，キャリッジリターン，またはラインフィードなどを含む場

合は，別の引用符，CHR$(34)によって囲んでファイルに書いてください。
例をあげると，A$＝〝CAMERA,□AUTOMATIC〟また，B$＝〝□□□93604－1〟のとき（□は空白を意味します），次の文，

PRINT #1，A$；B$

は次のようなイメージをファイルに書きます。

CAMERA,□AUTOMATIC□□□93604－1

そして次の文，

INPUT #1，A$, B$

は，〝CAMERA〟をA$に，そして〝AUTOMATIC93604－1〟をB$に読み込みます。これらの文字列をファイル上で正しく分割するには，CHR$(34)により引用符をファイルに書き込んで下さい。次の文，

PRINT # 1, CHR$(34)；A$ ；CHR$(34)； CHR$ (34)；B$；CHR$(34)

は次の内容をファイルに書き込みます。

〝CAMERA,□AUTOMATIC〟〝□□□93604－1〟

すると次の文，

INPUT #1，A$, B$

は〝CAMERA,□AUTOMATIC〟をA$に，〝□□□93604－1〟をB$に読み込みます。
PRINT #文は，USINGと共に使ってファイルのフォーマットを制御することができます。

**参照**　PRINT # USING，INPUT #，第1章 ファイル

# PRINT USING/LPRINT USING

**機 能** 文字列，数値を指定した書式で出力する．

**書 式** PRINT / LPRINT USING 〈書式制御文字列〉；〈式〉〔 ; / , …〕〔 ; / , 〕

**文 例** PRINT USING "#####,." ; A, B

**解 説** 〈書式制御文字列〉によって後に続く〈式〉の出力される領域や書式を決定します．

**数値の書式制御**

\# ………………数値を出力する桁数を指定します．指定した桁数より数値の桁数が小さいときには，右づめで出力されます．

. ………………小数点の位置を指定します．小数点以下の部分で冗長となる桁には0が出力されます．

＋………………〈書式制御文字列〉の最初または最後に付けた場合，数値の符号がそれぞれ前または後に出力されます．2個以上の"＋"を並べた場合には，余分は後述の制御文字以外の文字と同じ扱いになります．

－………………〈書式制御文字列〉の最後に付けた場合，数値が負の数の時に数値の後ろに"－"が出力されます．前に付けたり，2個以上並べた場合には後述の制御に文字以外の文字と同じ扱いになります．

＊………………〈書式制御文字列〉の先頭につけた場合，数値領域の左側に空白部分ができたとき，そこを"＊"で埋めて出力します．この"＊＊"は2桁分の領域を確保します．

￥￥……………〈書式制御文字列〉の先頭につけた場合，出力される数値の直前に"￥"を出力します．"￥￥"は2桁分の領域を確保しますが，このうち1桁分は"￥"の出力領域とし

て使われます．後述の指数形式の書式指定を行った場合には，正しく出力されない事があります．

＊＊¥…………〈書式制御文字列〉の先頭につけた場合，上記の二つ（＊＊と¥¥）の両方の機能となります．〝＊＊¥″は3桁分の領域を確保しますが，このうち1桁分は〝¥″の出力領域として使われます．

，………………桁数指定の〝#″の並びの中においた場合，数値の整数部が3桁毎に〝,″で区切られて出力されます．ただし，前記の〝.″より右側においた場合は，数値の最後に〝,″が出力され，3桁毎の区切りは行われません．

∧∧∧∧………桁数指定の〝#″の後に付けた場合，数値は指数形式で出力されます．

**制御文字以外の文字**

以上の制御文字以外の文字（英数字，カナ，グラフィック記号等）を置いた場合，数値の前や後ろにそのキャラクタが出力されます．

**数値の領域をこえた場合**

指定した数値領域より数値の桁数が大きい場合，数値の直前に〝%″が出力されます．数値の丸めが領域より大きくなる原因となった場合も，丸めた数値の前に〝%″が出力されます．

サンプルプログラム

```
10 REM ** USING **
20 PRINT "4 ｹﾀ ﾉ ﾗﾝｽｳ ｦ 200 ｺ ﾂｸﾘﾏｽ｡"
30 FOR I=0 TO 24
40 FOR J=0 TO 7
50 R=RND(1)*10000
60 PRINT USING"#####";R;
70 NEXT J,I
```

# PRINT♯ USING

機 能　文字列，数値を指定した書式でファイルに出力する．

書 式　PRINT #<ファイル番号>，USING <書式制御文字列> ; <式>
〔 ; , <式>…〕〔 ; , 〕

文 例　PRINT #2，USING "####" ; A

解 説　<ファイル番号>で指定されたファイルに対して，文字列や数値を書式指定して出力します．
PRINT # USING は，その対象がファイルであることを除けば PRINT USING とまったく同じです．

参 照　PRINT USING，PRINT #，OPEN

# PSET

機　能　　**画面上の任意のドットをセットする.**

書　式　　PEST (<水平座標>, <垂直座標>〔, <ファンクションコード>〕)

文　例　　PEST (80, 32)

解　説

指定した座標にドットをセットします. 〈水平座標〉, 〈垂直座標〉, 〈ファンクションコード〉は共に0から255の範囲でないとエラーになります. また, LCD画面のドット座標系は239×63となっており, 〈水平座標〉は239以上はすべて239として, 〈垂直座標〉は63以上はすべて63として処理されます. 既に表示されているキャラクタの上に重ねてドットをセットすることも可能です.

〈ファンクションコード〉に偶数 (0～254) を指定するとPSETの機能は反転し, PRESETとまったく同じ働きをします(ドットをリセットする). 奇数 (1～255) を指定した場合は, 省略した時と同様にドットのセットを行います.

CRTが接続されている場合は, パラメータの範囲や〈ファンクションコード〉の役割が異なっていますのでCRT付属のマニュアルを参照してください.

注　意

画面がスクロールするとドットグラフィクスはすべてクリアされます.
画面の座標は常に左上隅が (0, 0) です.

参　照　　PRESET, LINE

サンプルプログラム

```
10 REM *** PSET ***
20 SCREEN 0,0:CLS
30 A=150:B=.05:C=11
40 FOR T=-15 TO 72 STEP .13
50 X=EXP(-T*B)*COS(160*3.14*T/180-A)
60 Y=EXP(-T*B)*COS(160*3.14*T/180-C)
70 X=X*120+120:Y=Y*32+32
80 IF X>=0 AND X<256 AND Y>=0 THEN PSET(X,Y)
90 NEXT
100 BEEP
110 GOTO 110
```

# READ

機 能　DATA 文より値を読み，変数に割り当てる.

書 式　READ <変数>〔,<変数>…〕

文 例　READ K, M, S$

解 説　READ 文はいつでも DATA 文と組み合わせて使わなければなりません．READ 文は DATA 文のデータを，1対1対応の方法で変数に割りあてていきます．READ 文の変数は数値変数でも文字型変数でもかまいません．しかし読み出された値と変数の型は一致していなければなりません．もし，型が一致しない場合には，〝?SN Error〞(Syntax error)が起こります．

一つの READ 文が一つまたはそれ以上の DATA 文を参照したり（複数の場合は順番に参照されます)，またいくつかの READ 文が一つの DATA 文を参照することができます．もし<変数>の並びの数が DATA 文のデータの個数を越えてしまった場合は，〝?OD Error〞(Out of DATA)のエラーメッセージが表示されます．指定された変数が DATA 文のデータの個数よりも少ない場合には，その次の READ 文は読まれなかったデータから読み始めます．もしそれ以上 READ 文がない場合には，余分のデータは無視されます．

一度読んだ DATA 文を読みなおすには，RESTORE 文を使います．

参 照　RESTORE, DATA

サンプルプログラム

```
10 REM ** READ **
20 CLS:LOCATE 8,3
30 FOR I=0 TO 8
40 READ R$
50 PRINT R$;"  ";
60 NEXT
70 END
80 DATA Please,read,this,manual.,"
90 DATA I(PC-8201),am,reading,DATA.
```

# REM

**機　能**　プログラムに注釈を入れる.

**書　式**　REM〔〈注釈文〉〕

**文　例**　REM TESTPROGRAM

'TESTPROGRAM

**解　説**　REM 文は非実行文であり，プログラムの実行に影響を与えずコメント行とすることができます．リストを取ると入力した内容がそのまま出力されます．ただし，REM 文は GOTO 文，GOSUB 文の飛先として使うことは可能です．

REM 文ではキーワード〝REM〞の代わりにアポストロフィ〝'〞を使うことができます．REM 文の行はコロンで区切って他の文を続けることはできませんが，逆に他の文に続けてコロンで区切って REM 文を置くことはできます．

**サンプルプログラム**

```
10 REM ** REM **
20 REM ﾊ ﾌﾟﾛｸﾞﾗﾑ ﾆ ﾁｭｳｼｬｸ ｦ ﾂｹﾏｽ｡
30 '    ｱﾎﾟｽﾄﾛﾌｨ ﾃﾞﾓ ﾀﾞｲﾖｳ ﾃﾞｷﾏｽ｡
40 REM PC ﾊ REM ｲｶ ｦ ﾑｼｼﾏｽ｡
50 REM ｲｶ ﾆ ﾅﾆｶ ﾒｲﾚｲ ｦ ﾂｹﾃﾓ ﾑｼｻﾚﾏｽ｡
60 REM ﾀﾄｴﾊﾞ ･･ :Print "ﾑﾀﾞ"
70 PRINT" ｿﾉ ｷﾞｬｸ ﾊ ﾃﾞｷﾏｽ｡":REM ﾑﾀﾞ
```

# RENUM

| | |
|---|---|
| 機　能 | **プログラムの行番号を整理する.** |
| 書　式 | RENUM〔<新行番号>〕〔,<旧行番号>〕〔,<増分>〕 |
| 文　例 | RENUM |

**解　説**

<新行番号>は，新しくつける行番号の最初の行番号で，省略値は10です．<旧行番号>は行番号のつけ替えをはじめる現在のプログラムの行番号です．省略値はそのプログラムの最初の行番号です．<増分>は新しく付ける各行番号のあいだの増分で，省略値は10です．

RENUM コマンドはまた，GOTO，GOSUB，THEN，ON…GOTO，ON…GOSUB 及び ERL 文などで参照している行番号も新しい行番号に対応して変更します．これらの文が参照している行番号の行がもし存在しない場合には，〝Undefined line xxxx in yyyy〟とエラーメッセージが表示されます．この場合，誤った行番号（xxxx）は RENUM コマンドによって変更されませんが，行番号 yyyy は変更されます．

RENUM コマンドの実行終了後は，いつでもコマンドレベルに戻ります．

**注　意**

RENUM コマンドをプログラム行の順序を変えるのに使うことはできません(たとえば，10，20，30の三つの行がある場合の RENUM15，30 など)．また65529以上の行番号を発生することもできません．このような場合には〝？FC Error〟（Illegal function call エラー）が起こります．

# RESTORE

**機 能** READ 文で読む DATA 文を指定する.

**書 式** RESTORE 〔<行番号>〕

**文 例** RESTORE 800

**解 説** <行番号>が省略されると，次にくる READ 文はプログラム中の最初の DATA 文から読み始めます.<行番号>を指定すると，指定された行の DATA 文から読み始めます.

**サンプルプログラム**

```
10 REM ** RESTORE **
20 FOR I=0 TO 19
30 READ A$:PRINT A$;"  ";
40 RESTORE 80
50 NEXT I
60 RESTORE 90
70 READ A$:PRINT A$
80 DATA ﾅﾝﾄﾞﾃﾞﾓ
90 DATA "DATA ｶﾞ ﾖﾒﾏｽ｡"
```

# RESUME

**機　能**　エラー回復処理終了後，プログラムの実行を再開する．

**書　式**　RESUME ［0］
　　　　　　　　 NEXT
　　　　　　　　 <行番号>

**文　例**　RESUME 0
RESUME NEXT
RESUME 1000

**解　説**　プログラムの実行を再開する場所に応じて次のように三つの書式を選ぶことができます．

RESUME または RESUME 0
　エラーの原因となった文からプログラムの実行が再開されます．

RESUME NEXT
　エラーの原因となった文のすぐ次の文から実行が再開されます．

RESUME <行番号>
　<行番号>で指定した行から実行が再開されます．

**参　照**　ON ERROR GOTO

# RETURN

**機能**　サブルーチンから復帰する.

**書式**　RETURN〔〈行番号〉〕

**文例**　RETURN

RETURN 200

**解説**　GOSUB文によってコールされたサブルーチンの最後にRETURN文を入れる事によって,GOSUB文の次の文へ復帰し,プログラムの実行を続けます. 割り込みによるサブルーチン（ON COM GOSUB文）の場合, 割り込みが発生した時に実行し終った文の次の文へ復帰します.

サブルーチンは必ずGOSUB文によってコールされなければなりません. もし単独で実行したり, 誤ってGOTO文などで実行された場合, RETURN文に出会うとエラーになります. また, 一つのサブルーチン内に複数のRETURN文があってもかまいませんが, 正しくGOSUB文と対応していなければなりません.

なも, RETURN文の後に〈行番号〉を指定して, 特定の行へ強制的にリターンさせることもできます.

**注意**　サブルーチンの中にCLEAR文を置くと, RETURNで戻るべき場所もクリアされてしまい, エラーの原因になります.

**参照**　CLEAR, GOSUB, ON～GOSUB

# RUN

**機 能** プログラムの実行を開始する。
ファイルを BASIC エリアにロードし，そのプログラムを実行する。

**書 式** 1) RUN 〔<行番号>〕
2) RUN <ファイルディスクリプタ> 〔, R〕

**文 例** 1) RUN 100
2) RUN "TEST"

**解 説** 1) <行番号>を指定すると，その行から実行がはじまります。指定のない場合には，最も若い行番号の行から実行がはじまります。プログラムの実行が終ると BASIC はいつでもコマンドレベルに戻ります。

2) <ファイルディスクリプタ>で指定されるプログラムをロードし，その後プログラムの実行を開始します。デバイス名として "CAS：" を指定した場合はカセットから，省略した場合は "RAM：" とみなされ）本体 RAM 上からファイルをロードしてきます。その他オプションとしてフロッピィディスクを対象とした "<番号>：", 外部記憶装置を対象とした "0：" が用意されています。

RUN コマンドを実行すると，すべての開いているファイルを閉じてすべての変数をクリアし，目的のプログラムをロードする場合はまず BASIC エリアの内容をクリアします。しかし，R オプションを付けた場合には，すべてのデータファイルは開いたままになります。

**注 意** RUN "CAS：" を指定した場合，ファイルのロード中にその実行を中断したい時は，SHIFT + STOP キーを押さなければなりません。

**参 照** LOAD

**サンプルプログラム**

```
10 REM * RUN ハ PROGRAM デハ アマリ ツカワナイ *
20 PRINT "RUN シタラ トマラナイ PROGRAM"
30 PRINT
40 PRINT "STOP ヲ オシテ !"
50 PRINT
60 RUN "RUN2"

10 REM *** RUN 2 ***
20 PRINT "イマ RUN 2 ヲ ジッコウチュウ"
30 PRINT
40 PRINT "ツギ ハ RUN 1 ニ モドリマス "
50 PRINT
60 RUN "RUN1"
```

# SAVE

**機 能**　**プログラムを指定したファイル名でセーブする．**

**書 式**　SAVE "〈ファイルディスクリプタ〉"〔, A〕

**文 例**　SAVE "ENERGY", A

**解 説**　BASICエリア上のプログラムを指定されたファイル名（6文字以内）でセーブし，コマンドレベルに戻ります．〈ファイルディスクプリタ〉のデバイス名を省略すると "RAM：" が選択され，拡張子を省略するか ".BA"に指定すれば ".BA"ファイルを，拡張子を ".DO" に指定するかオプションの ",A" をつければ ".DO" ファイル（ASCII形式）を作ることができます．RAM上に同じファイル名を持つファイルが存在すると，そのファイルは新しいファイルに置き換えられてしまうので注意が必要です．

一度セーブしたファイルはまったく同じファイル名で別のプログラムをセーブするか，KILL命令を実行するか，コールドスタートしない限りファイルとして保存されます．また，同じファイル名で同じプログラムを2度続けてセーブしようとすると "?FC Error" になります．

デバイス名はその他，カセットテープを対象とした "CAS："，RS-232C回線を対象とした "COM："，プリンタを対象とした "LPT：" があります．"CAS：" についてはCSAVE，"COM：" はOPEN "COM："，"LPT：" はLLISTをそれぞれ参照してください．

オプションとして外部記憶装置用に "0："，フロッピィディスク用には "〈番号〉：" も用意されています．

SAVE実行後はいつでもコマンドレベルに戻ります．

**注 意**　RAM上にあるプログラムファイルは，一度LOAD命令によってBASICエリア上に置かなければセーブできません．SAVEの前にLISTを実行して，目的のプログラムがあることを確かめれば，勘違いを避けることができます．

プログラムをRAM上にセーブする場合，".BA" ファイルにするとLOADに時間がかからず，メモリもあまり使わなくて済みます．しかし，そのファイルにアクセスしたまま（FILESを実行した時アスタリスク "*" が示される）スクリーンエディットを行うと，ファイルそのもの

が書き換えられてしまいます．メモリに余裕があれば常に〝.DO″ファイルとしてセーブし，余裕がなければ〝.BA″ファイルとしてカセットにセーブするのが効率の良いセーブ方法の一つです．
またカセット上に〝.DO″（ASCII形式）ファイルを作る時はSAVE〝CAS：〈ファイル名〉″，Aのようにしてください．
SAVE〝CAS：″の実行を中断したい時は，SHIFT＋STOPキーを押さなければなりません．

参照　CSAVE, LOAD, LLIST, BSAVE, OPEN〝COM：″，第1章ファイル

# SCREEN

| | |
|---|---|
| 機　能 | **画面のモードを設定する.** |
| 書　式 | SCREEN <出力デバイス番号>〔,<ファンクションキー表示スイッチ>〕 |
| 文　例 | SCREEN 0,0 |

解　説　<出力デバイス番号>は0がLCD，1がCRTに対応し，<ファンクションキー表示スイッチ>は0が表示しない，1が表示することを意味します．ファンクションキーを表示しない場合，キャラクタ表示は8行まで可能になります．<出力デバイス番号>を省略する場合はコンマ〝,〟は省略できず，逆に<ファンクションキー表示スイッチ>を省略する場合は，〝,〟も省略しなければなりません．省略されたパラメータはそれまでの値を保持しますが，両方共省略するとエラーになります．

注　意　両パラメータ共，マイナスや256以上を指定するとエラーになります．また，CRTが接続されていない時にSCREEN 1を実行してもエラーになります．

参　照　CLS

サンプルプログラム

```
10 REM *** SCREEN ***
20 FOR I=0 TO 21
30 SCREEN 0,I MOD 2
40 NEXT
```

# SOUND

| 機 能 | 指定した音を鳴らす. |
|---|---|
| 書 式 | SOUND <音程>, <長さ> |
| 文 例 | SOUND 5586, 50 |

解 説　音程と長さを指定して音を出します. <音程>は 0 から16383までの整数で, 大きい程低い音が出ます. 5586を指定すれば440Hz の音が得られます. <長さ>は 0 から255までの整数で 1 単位の長さは0.02秒です.

**音階表**

| (コード) | 1 | 2 | 3 | 4 | 5 | 6 (オクターブ) |
|---|---|---|---|---|---|---|
| C | 低 | 9394 | 4697 | 2348 | 1171 | 587 |
| C# | | 8866 | 4433 | 2216 | 1103 | 554 |
| D | | 8368 | 4184 | 2092 | 1045 | 523 |
| D# | 15800 | 7900 | 3950 | 1975 | 987 | 493 |
| E | 14912 | 7456 | 3728 | 1864 | 932 | 466 |
| F | 14064 | 7032 | 3516 | 1758 | 879 | 439 |
| F# | 13284 | 6642 | 3321 | 1660 | 830 | 415 |
| G | 12538 | 6269 | 3134 | 1567 | 783 | |
| G# | 11836 | 5918 | 2954 | 1479 | 733 | 高 |
| A | 11172 | *5586 | 2793 | 1396 | 693 | |
| A# | 10544 | 5272 | 2636 | 1316 | 653 | |
| B | 9952 | 4968 | 2486 | 1244 | 622 | |

サンプルプログラム

```
10 REM ** SOUND **
20 DIM S(17):Z#=4697
30 FOR I=1 TO 17
40 S(I)=Z#
50 Z#=Z#/1.0594639#
60 NEXT
70 FOR I=1 TO 16
80 SOUND S(15),32/I:SOUND S(17),32/I
90 SOUND S(13),32/I:SOUND S(1),32/I
100 SOUND S(8),48/I:SOUND S(0),16/I
110 NEXT I
```

# STOP

**機　能**　**プログラムの実行を停止してコマンドレベルに戻る.**

**書　式**　STOP

**文　例**　STOP

**解　説**　STOP 文はプログラムの実行を停止するためのもので，プログラムのどこで使用してもかまいません．STOP 文を実行すると，次のメッセージがプリントされます．

　　Break in llll　(llll はストップした行番号)

END 文と異なり，STOP 文はファイルを閉じません．

STOP が実行されると，BASIC はコマンドレベルに戻ります．また，CONT コマンドによりプログラムの実行は再開されます．

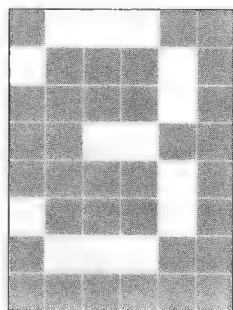
**参　照**　CONT

**サンプルプログラム**

```
10 REM ** STOP **
20 PRINT" STOP ﾊ DEBUG ﾆ ﾂｶｳ｡"
30 PRINT" ﾂﾂﾞｹﾙ ﾆﾊ CONT ｦ ﾂｶｳ｡"
40 STOP
50 I=I+1:PRINT I;"ｶｲﾒ ﾉ ｼﾞｯｺｳ ｻｲｶｲ｡"
60 GOTO 30
```

# WIDTH

| | |
|---|---|
| 機能 | CRT 画面の行数及び桁数を指定する. |
| 書式 | WIDTH <桁数>, <行数> |
| 文例 | WIDTH 80, 25<br>WIDTH 40<br>WIDTH ,20 |
| 解説 | CRT に出力するときの行数と桁数を指定します. 桁数は40と80, 行数は20と25だけが用意されています. パラメータの省略の方法は文例の通りです. |
| 注意 | WIDTH は CRT 以外のデバイスには使えません. |

# 関　数

# ABS

機 能　絶対値を与える.

書 式　ABS(<数式>)

文 例　A＝ABS(−1.6)

解 説　<数式>の絶対値を与えます.

サンプルプログラム

```
10 REM ** ABS **
20 CLS:SC=100:PRINT" ｱﾝｻﾞﾝ ﾉ ｾｲｶｸｻ ﾉ ﾃｽﾄ｡"
30 PRINT" ｱﾝｻﾞﾝ ﾉ ｾｲｶｸｻ ﾉ ﾃｽﾄ 10 ｶｲ｡"
40 FOR I=0 TO 9
50 R=INT(RND(1)*200):P=INT(RND(1)*200)
60 LOCATE 3,3:PRINT R;"ﾀｽ";P;"ﾊ"
70 FOR J=0 TO 500:NEXT J
80 LOCATE 3,3:PRINT SPACE$(20)
90 INPUT A:C=R+P:M=ABS(C-A)
100 IF A=C THEN PRINT"ｾｲｶｲ !" ELSE PRINT"ｺﾀｴ ﾊ";C;"ﾃﾞｼﾀ｡ ｹﾞﾝ
ﾃﾝ";M
110 FOR J=0 TO 300:NEXT J:CLS:SC=SC-M
120 NEXT I
130 PRINT" YOUR SCORE";SC
```

# AND

| 機能 | 2式の論理積を与える. |
|---|---|
| 書式 | 〈数式1〉 AND 〈数式2〉 |
| 文例 | IF A＝5 AND B＞2 THEN 200<br>PRINT 23130 AND A＋3 |

**解説**

通常IF文と共に用いて条件分岐に使われ，関数であることを意識する必要はありません．関数として使う場合，それぞれの数式を16桁の2進数として各ビットごとに論理積をとり，その結果である16桁の2進数を10進数に変換して返します．このため数式の値は −32768 から 32767 の整数でなければならず，小数部を含む場合は，切り捨てられます．論理積の計算は次のようになります．

1 AND 1 → 1
1 AND 0 → 0
0 AND 1 → 0
0 AND 0 → 0

16桁の場合

0101 0101 1111 1111 AND 0010 1110 0100 1101 → 0000 0100 0100 1101

これを実際の画面で試してみると

```
PRINT 11853 AND 22015
 1101
```

となります．

**注意**

負の値は2の補数で表現されるため，論理代数の知識が必要です．

**参照**

EQV，IMP，NOT，OR，XOR，第1章 演算子

# ASC

**機能**　**文字のキャラクタコードを与える.**

**書式**　ASC(<文字列>)

**文例**　A＝ASC("A")

**解説**　<文字列>の最初のキャラクタコードを与えます。

文字とキャラクタコードとの対応については,「キャラクタコード表」を参照して下さい。

<文字列>がヌルストリングの場合には"?FC Error"になります.

**参照**　CHR$,第7章キャラクタコード表

**サンプルプログラム**

```
10 REM *** ｱﾝｺﾞｳ ﾌｧｲﾙ ***
20 MAXFILES=1
30 CLS:INPUT" ｱﾝｺﾞｳ ｶｲﾄﾞｸ(K),ﾂｸﾙ(T)";Y$
40 IF Y$<>"K" AND Y$<>"T" THEN 30
50 INPUT" ｱﾝｺﾞｳ ﾌｧｲﾙﾒｲ(.DO ﾉﾐ!)";N$
60 IF Y$="K" THEN 130
70 OPEN N$ FOR OUTPUT AS #1
80 PRINT" ｱﾝｺﾞｳ ｦ ﾂｸﾘﾏｽ､ｽｷﾅ ﾌﾞﾝ ｦ ﾆｭｳﾘｮｸ ｼﾃｸﾀﾞｻｲ"
90 PRINT" ｵﾜﾘ ﾊ ESC ｷｰﾃﾞｽ｡"
100 X$=INKEY$:IF X$="" THEN 100
110 IF ASC(X$)=27 THEN CLOSE:GOTO 30
120 J=ASC(X$):PRINT #1,J;:GOTO 100
130 OPEN N$ FOR INPUT AS #1
140 PRINT" ｶｲﾄﾞｸ ﾊｼﾞﾒ !"
150 IF EOF(1) THEN CLOSE:GOTO 170
160 INPUT #1,X:PRINT CHR$(X);:GOTO 150
170 PRINT:PRINT "END HIT ANY KEY !"
180 IF INKEY$="" THEN 180 ELSE 30
```

# ATN

**機 能**　逆正接（アークタンジェント）を与える．

**書 式**　ATN(〈数式〉)

**文 例**　A＝ATN(0.5)

**解 説**　〈数式〉の逆正接をラジアンで与えます．
得られる値は，$-\pi/2$ から $\pi/2$ までの範囲です．

# CDBL

**機 能**　整数値，単精度実数値を倍精度実数値に変換した数値を与える．

**書 式**　CDBL(〈数式〉)

**文 例**　A#＝CDBL(B!／2)

**解 説**　〈数式〉の値を倍精度実数値に変換します．ただし，型変換が行われるだけで有効桁数の変化はありません．

**参 照**　CINT, CSNG

# CHR$

**機能** 指定したキャラクタコードに対応する文字を与える.

**書式** CHR$(〈数式〉)

**文例** A$=CHR$(65)

**解説** 〈数式〉の値のキャラクタコードに対応する文字を与えます. 〈数式〉の値が0〜255の範囲にない場合は, "?FC Error" が起こります.
〈数式〉には実数を含めてもかまいませんが, 小数点以下を切り捨てたものを〈数式〉の値とします.

**参照** ASC, 第7章キャラクタコード表

**サンプルプログラム**

```
10 REM *** CHR$ ***
20 FOR I=0 TO 28
30 READ C:PRINT CHR$(C);:NEXT
40 DATA 12,32,186,218,202,32,177,221
50 DATA 186,222,179,204,222,221,195,222
60 DATA 32,177,217,161,10,13,32,208,192
70 DATA 197,32,33,7
```

# CINT

**機　能**　単精度実数値，倍精度実数値を整数に変換した数値を与える．

**書　式**　CINT(〈数式〉)

**文　例**　A%=CINT(B#／2)

**解　説**　〈数式〉の値の小数点以下を切り捨てて整数値に変換します．
〈数式〉の値が−32768から32767の範囲にない場合は，〝?OV Error〟が起こります．

**参　照**　CDBL, CSNG, FIX, INT

# COS

**機　能**　余弦（コサイン）を与える．

**書　式**　COS(〈数式〉)

**文　例**　A=COS(3.1415926／2)

**解　説**　〈数式〉の値に対する余弦を与えます．
〈数式〉の単位はラジアンです．

**参　照**　SIN, TAN

# CSNG

| 機 能 | **整数値，倍精度実数値を単精度実数値に変換した数値を与える．** |
|---|---|
| 書 式 | CSNG(〈数式〉) |
| 文 例 | A!＝CSNG(B#) |

解 説　〈数式〉の値を有効数字6桁の単精度実数値に変換します．
〈数式〉の値が－1.70141 E＋38から1.70141 E＋38の範囲にない場合は，〝?OV Error〟が起こります．

参 照　CDBL, CINT

# CSRLIN

| 機 能 | **画面上でのカーソルの垂直位置を与える．** |
|---|---|
| 書 式 | CSRLIN |
| 文 例 | Y＝CSRLIN |

解 説　与えられる値は，画面上でのカーソルの垂直位置です．
値の範囲は，0から（画面の行数－1）です．

参 照　POS

# DATE$

|機 能| 日付を与える.

|書 式| DATE$

|文 例| D$=DATE$

PRINT DATE$

|解 説| DATE$には常に現在の年月日が入れられています．また，DATE$には，YY／MM／DD（"83／08／15" 等）の文字定数の形で日付をセットできます．他の文字型変数に代入した場合も，同じ形の文字列が代入されます．一度正しい日付をセットすれば後日あらためてセットする必要はありませんが，年については自動的に繰り上がることはないので年の変わる時には再度代入しなければなりません．またうるう年は考慮されていません．

|参 照| TIME$

# DSKF（ディスク）

|機 能| フロッピィディスクの残り容量をクラスタ単位で返す.

|書 式| DSKF（<ドライブ番号>）

|文 例| PRINT DSKF(1)

|解 説| <ドライブ番号>で指定されたフロッピィディスクの残り容量をクラスタ単位で返します.

|注 意| フロッピィディスク装置が接続されている時のみ使える関数です．フロッピィディスク装置付属のマニュアルも必ず参照してください．

# DSKI$ (ディスク)

| | |
|---|---|
| 機能 | **フロッピィディスクからの直接読み出しをする.** |
| 書式 | DSKI$ (<ドライブ番号>,<トラック番号>,<セクタ番号>,<スイッチ>) |
| 文例 | D$=DSKI$ (2,1,19,1) |

**解説** 通常のファイル操作(OPEN, CLOSEなど)とは無関係に，フロッピィディスク上の指定したセクタから直接読み出しを行います.

DSKI$は，そのパラメータにより指定されたセクタ上に書かれているデータ256バイトを読み出すとともに,<スイッチ>によって指定した前半または後半128文字の文字列を関数の値として返します.

文字列の長さは255文字までしか許されないために，関数の値として256バイトのデータすべてを得ることはできません.そこで，1セクタを2回に分けて読むために，<スイッチ>の指定が必要になります. <スイッチ>は0か1で指定し，0であると前半128バイトを，1の場合には後半128バイトを返します.

<トラック番号>, <セクタ番号>として指定できる値の範囲は，指定された<ドライブ番号>のディスクドライブの種類によって異なりますが，BASICは，これらの値の範囲を自動的に調べ，不当であった場合には〝Bad track/sector〞エラーとします.

**注意** フロッピィディスク装置が接続された時のみ使える関数です.

フロッピィディスク装置付属のマニュアルも必ず参照してください.

**参照** DSKO$, FIELD, OSKF, VARPTR

# EOF

**機能**　ファイルの終了コードを与える．

**書式**　EOF(〈ファイル番号〉)

**文例**　IF EOF(3) THEN CLOSE #1 ELSE GOTO 100

**解説**　〈ファイル番号〉で指定したファイルは入力モードでオープンされていなければなりません．

〈ファイル番号〉で指定したファイルが終わりに達したかどうかを調べます．終わりに達していれば真（−1），そうでなければ偽（0）を返します．

**サンプルプログラム**

```
10 REM *** EOF ***
20 OPEN"TSTEOF"FOR OUTPUT AS #1
30 INPUT"ﾃﾞｰﾀｦ ﾅﾝｶｲ ｶｷﾏｽｶ";N
40 FOR I=1 TO N
50 PRINT #1,I;
60 NEXT
70 CLOSE
80 OPEN"TSTEOF"FOR INPUT AS #1
90 IF EOF(1) THEN END
100 INPUT#1,N
110 PRINT N;
120 GOTO 90
```

# EQV

機 能　2式の論理演算（同値）の結果を与える．

書 式　<数式1> EQV <数式2>

文 例　B＝23＋A EQV 256

解 説　それぞれの数式の値を16桁の2進数として各ビットごとに論理演算（同値）を行い，その結果である16桁の2進数を10進数に変換して返します．このため数式の値は −32768 から 32767 の整数でなければならず，小数部を含む場合は切り捨てられます．

同値の計算は次のようになります．

1 EQV 1 → 1

1 EQV 0 → 0

0 EQV 0 → 1

0 EQV 1 → 0

16桁の場合

0101 1010 1111 0000 EQV 0010 1110 0100 1101 → 1000 1011 0100 0010

これを実際の画面で試してみると

PRINT 23280 EQV 11853

—29886

となります．

注 意　負の値は2の補数で表現されるため，論理代数の知識が必要です．

参 照　AND，IMP，NOT，OR，XOR，第1章 演算子

# ERL/ERR

| | |
|---|---|
| 機　能 | 発生したエラーコード及びエラーの発生した行番号を与える. |
| 書　式 | ERL<br>ERR |
| 文　例 | A＝ERL<br>B＝ERR |

解　説　エラーが発生した時点で，ERR はエラーコードを，ERL はエラーの発生した行番号を持っています．エラーがダイレクトモードの実行によって生じた場合，ERL は行番号として65535を持ちます．
一般に，ERL 及び ERR は ON ERROR GOTO 文によって指定したエラー処理ルーチンにおいて，処理の流れを制御するために使われます．
ERL と ERR は予約変数なので，値を代入することはできません．

参　照　第2章 ON ERROR GOTO ，ERROR

# EXP

| | |
|---|---|
| 機　能 | e を底とした指数関数の値を与える. |
| 書　式 | EXP(<数式>) |
| 文　例 | A＝EXP(1) |

解　説　<数式>の指数関数の結果を値とします．
<数式>の値が87.33655より大きい場合には“?OV Error”が起こります．

# FIX

| | |
|---|---|
| 機　能 | **整数部を与える.** |
| 書　式 | FIX(〈数式〉) |
| 文　例 | A＝FIX(－B／3) |

解　説　〈数式〉の値の小数点以下を取り去った値を与えます.

参　照　INT, CINT

FIX(-10.5) = -10
(INT(-10.5) = -11)

# FRE

| | |
|---|---|
| 機　能 | **メモリの未使用領域の大きさを与える.** |
| 書　式 | FRE(〈式〉) |
| 文　例 | PRINT FRE(0)<br>PRINT FRE(A$) |

解　説　〈式〉が数式の場合は, BASIC の未使用テキスト領域のバイト数を与えます. また, 文字式の場合は, BASIC の未使用文字領域のバイト数を与えます.
〈式〉は単にダミーですから, 〈数式〉あるいは〈文字式〉ならば何であってもかまいません.
表示される未使用領域の大きさは, 変数やスタックなど BASIC プログラムの実行のための作業用領域を含んでいるため, 表示された大きさはあくまでも目やすとしてください.

# IMP

| 機能 | 2式の論理演算（包含）の結果を与える. |
|---|---|
| 書式 | 〈数式1〉 IMP 〈数式2〉 |
| 文例 | B＝23＋A　IMP 256 |

解説　それぞれの数式の値を16桁の2進数として各ビットごとに論理演算（包含)を行い，その結果である16桁の2進数を10進数に変換して返します. このため数式の値は －32768 から 32767 の整数でなければならず，小数部を含む場合は切り捨てられます.

包含の計算は次のようになります.

1 IMP 1 → 1
1 IMP 0 → 0
0 IMP 0 → 1
0 IMP 1 → 1

16桁の場合

0101 1010 1111 0000 IMP 0010 1110 0100 1101 → 1010 1111 0100 1111

これを実際の画面で試してみると

PRINT 23280 IMP 11853
—20657

となります.

注意　負の値は2の補数で表現されるため，論理代数の知識が必要です.

参照　AND，EQV，NOT，OR，XOR，第1章 演算子

# INKEY$

**機　能**　キーが押されていればその文字を，キーが押されてなければヌルストリングを与える．

**書　式**　INKEY$

**文　例**　A$=INKEY$

**解　説**　キーボードバッファが空であれば，INKEY$関数はヌルストリングを返します．キー入力によりキーボードバッファが空でない場合には，バッファの先頭から1文字取り出し，その文字を返します．なお，シフトキーなどの「キャラクタコード表」にないキーは無視されます．

**参　照**　第7章 キャラクタコード表

**サンプルプログラム**

```
10 REM ** INKEY$ **
20 SCREEN 0,0:CLS:X=20:Y=3
30 PRINT" U=UP,D=DOWN,R=RIGHT,L=LEFT"
40 PRINT" HIT ANY KEY"
50 A$=INKEY$:IF A$="" THEN 50
60 LOCATE X,Y:PRINT" ";
70 IF A$="U" AND Y>0 THEN Y=Y-1
80 IF A$="D" AND Y<7 THEN Y=Y+1
90 IF A$="R" AND X<39 THEN X=X+1
100 IF A$="L" AND X>0 THEN X=X-1
110 LOCATE X,Y:PRINT"X";
120 GOTO 50
```

# INP

**機　能**　入力ポートから値を得る。

**書　式**　INP(<ポート番号>)

**文　例**　A＝INP(15)

**解　説**　<ポート番号>で指定された入力ポートから8ビットのデータを読み取り，それを関数値とします。<ポート番号>として指定できる値の範囲は0から255までです。

**参　照**　OUT

# INPUT$

機能　指定されたファイルより指定された長さの文字列を与える．

書式　INPUT$(〈文字数〉〔,〔#〕〈ファイル番号〉〕)

文例　A$＝INPUT$(5,#3)

解説　〈ファイル番号〉によって指定されたファイルから，〈文字数〉分の文字列を読み出します．〈ファイル番号〉が省略された場合には，キーボードから入力されますが，INPUT文と異なり，入力された文字は画面に表示されません．

INPUT$は指定された〈文字数〉の文字が入力されるまで待ち続けますが，すでに入力バッファに入力済みのデータがある場合にはバッファ中から文字を拾ってきます．なお，INPUT$は STOP キーを除くすべての文字をそのまま読み込みますので，INPUT文やLINE INPUT文では入力することのできない ↵ (キャラクタコード13) 等も入力することができます．

サンプルプログラム

```
10 REM ** INPUT$ **
20 CLS:INPUT"PASSWORD";PW$
30 WL=LEN(PW$)
40 REM ** ﾎﾝﾄ ﾊ ｺｺｶﾗ ﾂｶｳ **
50 CLS:PRINT"PASSWORD:";
60 N$=INPUT$(WL)
70 IF N$=PW$ THEN PRINT" WELCOME USER !":SOUND 3000,20:GOTO
50
80 LOCATE 0,3:PRINT" UNIDENTIFIED PERSON USING PC 8201 !"
90 SOUND 5000,4:SOUND 1000,4
100 CLS:GOTO 80
```

# INSTR

| | |
|---|---|
| 機　能 | **文字列の中から任意の文字列を捜して，その文字列の位置を与える．** |
| 書　式 | INSTR(〔<数式>,〕<文字列1>,<文字列2>) |
| 文　例 | B＝INSTR(A$, "XYZ") |

解　説

〈文字列1〉の中から〈文字列2〉を捜し，発見した位置を値として返します．ただし，発見できない場合は0を返します．

〈数式〉は捜し始める位置で1～255の整数で指定します．これを省略すると〈文字列1〉の最初から捜し始めます．

〈文字列2〉にヌルストリングを指定すると，〈数式〉と同じ値を返しまが，〈数式〉の値が〈文字列1〉の長さを越える場合は0を返します．

サンプルプログラム

```
10 REM **** INSTR *****
20 DIM T(16)
30 FOR I=0 TO 16
40 READ T(I):NEXT
50 A$="ZSXCFVGBNJMK,L./"
60 B$=INPUT$(1)
70 SOUND T(INSTR(A$,B$)),10
80 GOTO 60
1000 DATA 12583,11836,11172,10544,9952,9394,8866,8368,7900
1010 DATA 7456,7032,6642,6269,5918,5586,5272,4968,4697
```

# INT

機能　小数点以下を切り捨てた整数値を与える.

書式　INT(〈数式〉)

文例　PRINT INT(－B／3)

解説　〈数式〉の値を超えない最大の整数を与えます.

参照　FIX, CINT

サンプルプログラム

```
10 REM *** INT ***
20 PRINT "   I         INT    FIX"
30 FOR I=-1.5 TO 1.6 STEP .3
40 PRINT USING"###.##  #####  #####";I,INT(I),FIX(I)
50 NEXT
```

# LEFT$

| | |
|---|---|
| 機　能 | **文字列の左側から任意の長さの文字列を与える.** |
| 書　式 | LEFT$(〈文字列〉,〈数式〉) |
| 文　例 | B$=LEFT$(A$, 4) |

解　説

〈数式〉の値は 0 から255の範囲になければなりません.〈数式〉が〈文字列〉の総文字数以上の場合は,〈文字列〉のすべてを結果とします.〈数式〉が 0 ならば, ヌルストリングを結果とします.

参　照

RIGHT$, MID$

サンプルプログラム

```
10 REM ** ﾎﾞｳｸﾞﾗﾌ **
20 A$="++++++++++++++++++++++++++++++++++++++++"
30 PRINT " ｸﾞﾗﾌ ﾉ ﾃﾞｰﾀ 6 ﾂ (0-39)"
40 FOR I=0 TO 5:PRINT I;
50 INPUT "ﾊﾞﾝﾒ ﾉ ﾃﾞｰﾀ";A(I)
60 IF A(I)<0 OR A(I)>39 THEN BEEP:PRINT "Illegal":PRINT I;:GOTO 50
70 NEXT I
80 FOR I=0 TO 5
90 PRINT LEFT$(A$,A(I))
100 NEXT I
```

# LEN

**機能** 文字列の総文字数を与える.

**書式** LEN(<文字列>)

**文例** A＝LEN(A$)

**解説** <文字列>にコントロールコードやヌルキャラクタが含まれていても出力されませんが，文字として数えられます.

**参照** 第7章 コントロールコード表

**サンプルプログラム**

```
10 REM ** LEN **
20 INPUT"36 ｼﾞｲﾅｲ ﾉ ﾓｼﾞﾚﾂ";N$
30 CLS:L=LEN(N$):GOSUB 60
40 PRINT "+ ";N$;" +"
50 GOSUB 60:END
60 FOR I=1 TO L+4
70 PRINT "+";:NEXT
80 PRINT:RETURN
```

# LOC

機　能　　ファイル中での論理的な現在位置を与える.

書　式　　LOC（<ファイル番号>)

文　例　　LAST＝LOC (2)

解　説　　<ファイル番号>で指定されたファイルが，オープンされてから，読み書きされたレコード数を返します.

注　意　　フロッピィディスク装置が接続された時のみ使える関数です.
フロッピィディスク装置付属のマニュアルも必ず参照してください.

# LOG

機　能　　自然対数を与える.

書　式　　LOG(<数式>)

文　例　　PRINT LOG(2.7182818)

解　説　　<数式>によって与えられた値の自然対数を返します. <数式>の値は 0 より大きくなければなりません.

# LPOS

**機 能**　**現在のプリンタバッファ中のプリンタヘッドの位置を与える.**

**書 式**　LPOS(<式>)

**文 例**　P＝LPOS(0)

**解 説**　式はダミーの引数で，変数か定数であれば何であってもかまいません。与えられる値はバッファ内のプリンタヘッドの位置で，必ずしも物理的なプリンタヘッドの位置を与えるとは限りません。

**参 照**　POS

# MID$

| 機　能 | 文字列の中から任意の長さの文字列を与える. |
|---|---|
| 書　式 | MID$（〈文字列〉,〈式1〉〔,〈式2〉〕） |
| 文　例 | B$=MID$（A$, 2, 3） |

**解　説**

〈文字列〉の〈式1〉番目から，〈式2〉個の文字を与えます．〈式1〉は1から255の範囲，また，〈式2〉は0から255の範囲になければなりません．
〈式2〉を省略した場合や，〈文字列〉の〈式1〉番目の文字から右の文字数が〈式2〉より小さい場合は，〈文字列〉の〈式1〉番目より右にあるすべての文字列を結果とします．
〈文字列〉の文字数が,〈式1〉より小さければ, MID$関数はヌルストリングを返します．

**参　照**　LEFT$, RIGHT$

**サンプルプログラム**

```
10 REM *** MID ***
20 A$="0123456789ABCDEF"
30 CLS
40 LOCATE 5,1:INPUT "ｾｲｽｳ";A:CLS
50 IF A<0 OR A>32767 THEN BEEP:GOTO 40
60 LOCATE 10,2:PRINT"ｼﾞｭｯｼﾝ ﾉ";A;"ﾊ"
70 LOCATE 10,4:PRINT"ｼﾞｭｳﾛｸｼﾝ ﾉ       ﾃﾞｽ"
80 N=0
90 H$=MID$(A$,(A MOD 16)+1,1)
100 A=INT(A/16)
110 LOCATE 24-N,4:PRINT H$
120 N=N+1
130 IF A=0 THEN 40
140 GOTO 90
```

# MOD

**機　能**　剰余を与える.

**書　式**　〈数式1〉 MOD 〈数式2〉

**文　例**　PRINT A MOD 7

**解　説**　数式は共に 32767 以内の正の整数でなければなりません. もし負の値を用いると絶対値として処理されますが,〈数式1〉に負の値を用いた場合, 結果を負の値として返します. また〈数式2〉には0を用いることはできません. また値が小数部を含む場合は切り捨てられます.

**注　意**　MOD は演算子としては+, -および関係演算子より上位にあるため, 数式がこれらの演算子を含む場合はカッコ"( )"で囲む必要があります.

**サンプルプログラム**

```
10 REM **** MOD ****
20 SCREEN 0,0:CLS
30 LOCATE 5,0:BEEP:INPUT"ｽｳｼﾞ";A:A=INT(A)
40 IF A<32768! THEN 60
50 PRINT"ｵｵｷｽｷﾞﾏｽ":FOR I=0 TO 1000:NEXT:GOTO 20
60 CLS:LOCATE 10,2:PRINT"ｼﾞｭｯｼﾝｽｳﾉ";A;"ﾊ"
70 LOCATE 10,4:PRINT"ﾆｼﾝｽｳ ﾃﾞﾊ"
80 N=0
90 LOCATE 30-N*2,6
100 PRINT A MOD 2;:A=INT(A/2):N=N+1
110 IF A<>0 THEN 90
120 GOTO 30
```

# NOT

| | |
|---|---|
| 機　能 | **式の論理演算（否定）の結果を与える．** |
| 書　式 | NOT<数式> |
| 文　例 | A＝NOT Q |

**解　説**　与えられた数式の値を16桁の２進数として各ビットを反転し（否定），その結果である16桁の２進数を10進数に変換して返します．このため数式の値は －32768 から 32767 の整数でなければならず，小数部を含む場合は切り捨てられます．否定の計算は次のようになります．

NOT　1　→　0

NOT　0　→　1

16桁の場合

NOT　0101 1010 1111 0000 → 1010 0101 0000 1111

これを実際の画面で試してみると

PRINT NOT 23280

−23281

となります．

**注　意**　負の値は２の補数で表現されるため，論理代数の知識が必要です．

**参　照**　AND，EQV，IMP，OR，XOR，第１章 演算子

# OR

機 能　2式の論理和を与える.

書 式　<数式1> OR <数式2>

文 例　IF A＝5 OR B＞2 THEN 200

PRINT 23130 OR A＋3

解 説　通常IF文と共に用いて条件分岐に使われ，関数であることを意識する必要はありません．関数として使う場合，それぞれの数式を16桁の2進数として各ビットごとに論理和をとり，その結果である16桁の2進数を10進数に変換して返します．このため数式の値は −32768 から 32767 の整数でなければならず，小数部を含む場合は切り捨てられます．論理和の計算は次のようになります．

1 OR 1 → 1

1 OR 0 → 1

0 OR 1 → 1

0 OR 0 → 0

16桁の場合

0101 1010 1111 0000 OR 0010 1110 0100 1101 → 0111 1110 1111 1101

これを実際の画面で試してみると

```
PRINT 23280 OR 11853
 32509
```

となります．

注 意　負の値は2の補数で表現されるため，論理代数の知識が必要です．

参 照　AND，EQV，IMP，NOT，XOR，第1章 演算子

# PEEK

**機　能**　メモリ上の指定された番地の内容を読み出す．

**書　式**　PEEK(〈番地〉)

**書　式**　A＝PEEK(61400)

**解　説**　〈番地〉によって指定されたメモリ番地の内容を与えます．〈番地〉は，0～65535の値でなければなりません．〈番地〉が小数を含む場合は，小数点以下を切り捨てます．
結果として1バイトの整数が与えられます．

**参　照**　第2章 POKE

# POS

**機　能**　現在のカーソルの水平位置を与える．

**書　式**　POS(〈式〉)

**文　例**　P＝POS(0)

**解　説**　式はダミーの引数で，変数か定数であれば何であってもかまいません．
与えられる値は，ディスプレイ上の現在のカーソルの水平位置(0～39)です．

**参　照**　CSRLIN

# RIGHT$

**機能** 文字列の右側から任意の長さの文字列を与える.

**書式** RIGHT$（<文字列>, <数式>）

**文例** PRINT RIGHT$（"ABCD", 3）

**解説** <数式>の値は 0 から255の範囲になければなりません.
<数式>が, <文字列>の総文字数以上の場合は, <文字列>のすべてを結果とします.
<数式>が, 0 ならばヌルストリングを結果とします.

**参照** LEFT$, MID$

**サンプルプログラム**

```
10 REM *** RIGHT$ ***
20 CLS:INPUT "ﾓｼﾞﾚﾂﾗ ｲﾚﾃ ｸﾀﾞｻｲ";A$:CLS
30 A$=RIGHT$(SPACE$(40)+A$,40)
40 B$=LEFT$(A$,1)
50 C$=RIGHT$(A$,39)
60 A$=C$+B$
70 LOCATE 0,4:PRINT A$
80 GOTO 40
```

# RND

**機能**　乱数を与える.

**書式**　RND(〈引数〉)

**文例**　A＝RND(1)

**解説**　0以上1未満の乱数を与えます. 発生する乱数は〈引数〉の値によって次のように異なります.

○負の場合——新しい乱数系列を設定します.

○0の場合——1つ前に発生した乱数の値をとります.

○正の場合——次の乱数を発生します.

与えられる乱数は0〜0.999999の実数です. コールドスタートしない限り初期乱数系列の先頭に戻ることはありません.

**サンプルプログラム**

```
10 REM *** RND ***
20 X=120:Y=32
30 SCREEN 0,0:CLS
40 X=X+INT(RND(1)*3)-1
50 IF X<0 OR X>255 THEN X=120
60 Y=Y+INT(RND(1)*3)-1
70 IF Y<0 OR Y>63  THEN Y=32
80 PSET(X,Y)
90 GOTO 40
```

0〜n-1 の乱数 INT(RND(1)*n)

1〜n の乱数 INT(RND(1)*n+1)

0〜D(n-1) の D おきの整数乱数 D*INT(RND(1)*n)

(例) 1と-1の時 D=2 n=2

N=2*INT(RND(1)*2)-1

# SGN

**機能**　符号を与える.

**書式**　SGN(〈数式〉)

**文例**　B=SGN(A)

**解説**　〈数式〉が正の場合は1を，〈数式〉が0の場合は0を，〈数式〉が負の場合には，−1を与えます.

**サンプルプログラム**

```
10 REM *** SGN ***
20 CLS:SCREEN 0,0
30 LOCATE 5,2:PRINT "1 ｶﾗ 100 ﾏﾃﾞﾉ ｶｽﾞ ｦ ｱﾃﾃ ｸﾀﾞｻｲ"
40 N=INT(RND(1)*100)
50 LOCATE 10,4:PRINT"          "
60 LOCATE 5,4:INPUT"ｲｸﾂ ﾄ ｵﾓｳ";A
70 LOCATE 10,6
80 ON SGN(N-A)+1 GOTO 100,150
90 PRINT "ﾓｯﾄ ｽｸﾅｲｮ":GOTO 50
100 FOR I=0 TO 4
110 LOCATE 10,6
120 BEEP:PRINT "    ｱﾀﾘ!!!    "
130 NEXT
140 GOTO 20
150 PRINT "ﾓｯﾄ ｵｵｷｲｮ":GOTO 50
```

# SIN

機 能　正弦（サイン）を与える．

書 式　SIN(<数式>)

文 例　PRINT SIN(3.14159／2)

解 説　<数式>の値に対する正弦を与えます．<数式>の単位はラジアンです．

参 照　COS，TAN

サンプルプログラム

```
10 REM *** SIN ***
20 SCREEN 0,0:CLS
30 X=0:N=0:F=1
40 Y=SIN(N/25)*32+33
50 PSET(X,Y)
60 IF X<1 THEN F=1
70 IF X>239 THEN F=-1
80 X=X+F:N=N+1
90 GOTO 40
```

# SPACE$

**機能** 任意の長さの空白文を与える.

**書式** SPACE$(〈数式〉)

**文例** A$= "A"+SPACE$ (10)+ "C"

**解説** 〈数式〉の数だけの空白を持った文字列を与えます.
〈数式〉の値は 0 から255までの範囲になければなりません.

**参照** TAB

# SQR

**機能** 平方根（ルート）を与える.

**書式** SQR(〈数式〉)

**文例** A=SQR(2)

**解説** 〈数式〉の値に対する平方根を与えます.〈数式〉の値は 0 以上でなければなりません.

# STR$

| | |
|---|---|
| 機　能 | 数値を表す文字列を与える. |
| 書　式 | STR$(〈数式〉) |
| 文　例 | A$＝STR$(123) |

解　説　〈数式〉によって指定された値を文字列に変換します.

〈数式〉にはすべての型の数値が使えます.

参　照　VAL, STRING$

サンプルプログラム

```
10 REM ** STR$ **
20 PRINT" 1 ﾏﾀﾊ 2 ｹﾀ ﾃﾞﾆｭｳﾘｮｸ ｼﾃｸﾀﾞｻｲ"
30 INPUT"ｲﾏ ﾅﾝｼﾞ";H:H$=MID$(STR$(H),2)
40 IF LEN(H$)=1 THEN H$="0"+H$
50 INPUT" ﾅﾝﾌﾟﾝ ";M:M$=MID$(STR$(M),2)
60 IF LEN(M$)=1 THEN M$="0"+M$
70 INPUT" ﾅﾝﾋﾞｮｳ";S:S$=MID$(STR$(S),2)
80 IF LEN(S$)=1 THEN S$="0"+S$
90 TIME$=H$+":"+M$+":"+S$
100 PRINT "ｼﾞｶﾝ ｦ ";TIME$;" ﾆ ｾｯﾄ ｼﾏｼﾀ｡"
```

# STRING$

**機 能**　任意の文字を任意の数だけ与える.

**書 式**　STRING$（〈数式〉, |〈文字式〉| ）
　　　　　　　　　　　　　 |〈数式〉|

**文 例**　PRINT STRING$（10,65）

**解 説**　〈文字式〉または〈数式〉で指定された文字が，〈数式〉で与えられた字数だけ連なった文字列を返します．与える文字が〈文字列〉の場合，その文字列の最初の1文字が有効となります．〈数式〉の場合，その値をキャラクタコードとみなします．
〈数式〉の値は，0から255の範囲に限られます．

**参 照**　STR$

**サンプルプログラム**

```
10 REM ** STRING$ **
20 R=RND(1)*26+65:L=RND(1)*40
30 PRINT STRING$(L,CHR$(R))
40 GOTO 20
```

# TAB

**機　能**　現在のカーソルの示す行の任意の位置まで空白を出力する.

**書　式**　TAB(〈数式〉)

**文　例**　PRINT TAB(10) ; "ABC"

**解　説**　TAB は PRINT 文や LPPINT 文などの出力文中のみで使用されます.
現在のカーソルのある位置から, 画面の左端から〈数式〉で指定される位置（画面の左端から数える）まで空白を出力します.
〈数式〉の値は 0 から255（1が左端）まで指定することができます. また〈数式〉で指定された位置が, 現在のカーソルの位置より左側であった場合, 空白は出力せずカーソルの位置も変わりません.
SPACE$関数との区別に注意してください.

**参　照**　SPACE$

**サンプルプログラム**

```
10 REM *** TAB ***
20 FOR I=1 TO 21 STEP 4
30 PRINT STRING$(I,"#");TAB(22-I);"*"
40 NEXT
```

# TAN

機 能　正接（タンジェント）を与える．

書 式　TAN(<数式>)

文 例　A=TAN(3.1416／4)

解 説　<数式>の値に対する正接を与えます．<数式>の単位はラジアンです．

参 照　COS，SIN

# TIME$

機 能　内蔵クロックの時刻を与える．

書 式　TIME$

TIME$=〝HH : MM : SS〞

文 例　PRINT TIME$

解 説　TIME$は常に現在の時刻が入れられています．時刻セットするときHHは00から23まで，MMとSSは00から59までの数字文字列です．一度正しい時刻をセットすれば後日あらためてセットする必要はありません．

参 照　DATE$

# VAL

| | |
|---|---|
| 機　能 | 文字列の表す数値を与える． |
| 書　式 | VAL(<文字列>) |
| 文　例 | A=VAL("−123") |

**解　説**　<文字列>の最初の1文字が+，−，.，または数字でない場合には，関数値は0になります．

数字以外の文字が現れた場合には，それ以降の文字は無視されます．また，<文字列>の途中のスペースは無視します．

**参　照**　STR$，CHR$

**サンプルプログラム**

```
10 REM ***** VAL *****
20 CLS
30 IF S=VAL(RIGHT$(TIME$,2)) THEN 30
40 S=VAL(RIGHT$(TIME$,2))
50 M=VAL(MID$(TIME$,4,2))
60 T=VAL(TIME$)
70 LOCATE 10,3:PRINT TIME$
80 IF M<>0 OR S<>0 THEN 100
90 FOR I=1 TO T:SOUND 693,10:SOUND 1396,10:NEXT:GOTO 30
100 IF (S MOD 10)=0 THEN SOUND 11172,20:GOTO 30
110 IF (S MOD 5)=0 THEN SOUND 5586,10:GOTO 30
120 SOUND 2793,5:GOTO 30
```

# XOR

**機　能**　2式の排他的論理和を与える.

**書　式**　〈数式1〉 XOR 〈数式2〉

**文　例**　B=23130 XOR A+3

**解　説**　それぞれの数式を16桁の2進数として各ビットごとに排他的論理和をとり，その結果である16桁の2進数を10進数に変換して返します．このため数式の値は−32768から32767の整数でなければならず，小数部を含む場合は切り捨てられます．排他的論理和の計算は次のようになります．

1　XOR　1　→　0
1　XOR　0　→　1
0　XOR　1　→　1
0　XOR　0　→　0

16桁の場合

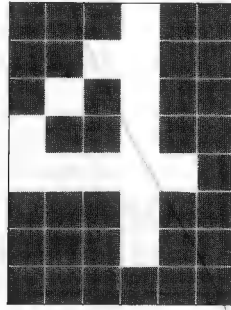
0101 1010 1111 0000　XOR　0010 1110 0100 1101　→　0111 0100 1011 1101

これを実際の画面で試してみると

```
PRINT 23280 XOR 11853
 29885
```

となります

**注　意**　負の値は2の補数で表現されるため，論理代数の知識が必要です．

**参　照**　AND，EQV，IMP，NOT，OR，第1章 演算子

# 機械語プログラム,及びキャラクタ定義

# 4 機械語プログラム,及びキャラクタ定義

## 4.1 機械語プログラムについて

機械語プログラムそのものについては，本書ではとても説明しきれるものではありません。ここでは $N_{82}$-BASIC で用意されている，機械語プログラムとのリンク機能を解説します。機械語プログラムは一歩間違えると暴走し，RAM 上のファイルやワークエリアを破壊したり，PC-8201が制御できなくなったりします．大切なプログラムやファイルは事前にカセットなどに保存しておく必要があります。ただしどんなに暴走したところでコンピュータそのものがこわれることはなく，コールドスタートすれば購入したときの状態（プログラムやファイルが〝BASIC〞，〝TEXT〞，〝TELCOM〞以外は何も入っていない）に戻すことができます．

ここからの説明を正しく理解するためには機械語プログラムに関する知識を必要とします．

### 4.1.1 機械語プログラムの準備

機械語プログラムはシステムワークエリア（62336番地以降）を除く RAM 上に用意します。初期設定ではBASICが，62335番地までをすべて使用するようになっているので，機械語プログラムに必要な領域は CLEAR 文の第 2 パラメータを指定して確保しなければなりません。BASIC で使う領域を機械語プログラムの領域として使用するとBASICが誤動作したり，BASICの動きによって機械語プログラムが書き換えられたりしてしまいます．

### 4.1.2 機械語プログラムのロード

機械語プログラムが〝.CO〞ファイルとして準備されている場合，BLOAD 命令によってこれを RAM 上にロードします．この時もし，その領域を CLEAR 文で確保していないと〝? OM Error〞エラーになり，BASIC 領域を不要意に破壊してしまうのを防ぎます．

BLOAD してくる〝.CO〞ファイルは通常 BSAVE 命令で作りますが，この時に実行開始番地の指定付にしておいた〝.CO〞プログラムファイルはロード終了と同時に実行を開始します（第 2 章BLOAD，BSAVEを参照)．実行開始後はEXECの実行後と同様になります。

### 4.1.3 機械語プログラムの書き込み

機械語プログラムを直接RAM上に書き込むにはPOKE文を使います．POKE文はCLEAR文で指定した範囲外の番地でも，値を書き込んでしまうので注意が必要です．

### 4.1.4 機械語プログラムの内容確認及び実行後の値の受け取り

指定した番地の内容を知るためにはPEEK関数を使います．後述のEXECで，A，H，L各レジスタの値をやりとりするときなどにPOKE文と組み合わせて使います．

### 4.1.5 機械語プログラムの実行

EXEC文を使って指定した番地からの機械語プログラムをサブルーチンとして実行し，機械語命令のリターンでBASICに復帰します．値を渡せるのは次のレジスタで，それぞれ指定した番地にPOKE文で値（1バイト）をEXEC文の実行前に書き込んでおきます．

Aレジスタ　63911番地
Lレジスタ　63912番地
Hレジスタ　63913番地

結果はやはり同じ番地に格納されるので，BASICに復帰してからこれらの番地の内容をPEEK関数で調べればサブルーチンの実行結果の値を受けとることができます．その他のレジスタに値を渡す場合は，機械語プログラム領域内にユーザーがワークエリアを決めてPOKE文，PEEK関数を使うことになります．

機械語サブルーチンが呼び出されたときは，スタック領域として8レベル（16バイト）使えるようになっています．これ以上スタックを使用するとBASICのワークエリアを破壊することになり，戻ってからの動作が保証できません．そこで，機械語サブルーチンがさらに広いスタック領域を必要とする場合は，BASICのスタックをセーブして，ユーザーのスタック領域を設定してください．たとえば次のようになります．

```
LXI H, 0
DAD SP
SHLD SSAVE    (SSAVEにBASICのスタックをセーブ)
  :           (ユーザルーチン)
  :
LHLD SSAVE    (BASICのスタックを回復)
SPHL
RET
```

### 4.1.6 機械語プログラムのセーブ

機械語プログラムのセーブには BSAVE 命令を使います．詳しいことは第2章の BSAVE を参照してください．

### 4.1.7 サンプルプログラム

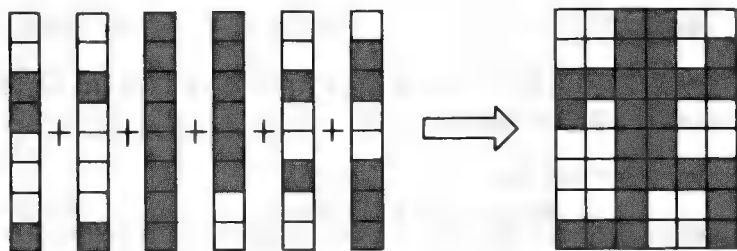
機械語プログラムの書き込みから実行までの様子をごく簡単な例で示します．このプログラムは入力した数値に1を加えて返すだけのものです．

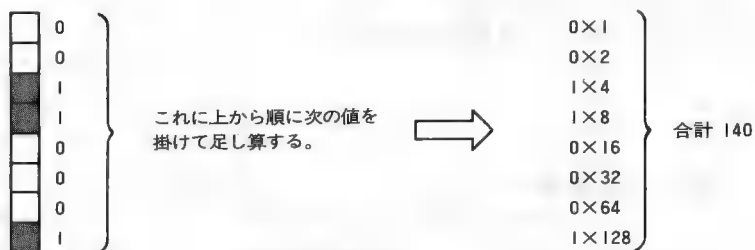
```
10 REM *** MACHINE INC ***
20 CLEAR 256,62330!
30 REM *** MACHINE SUB POKE ***
40 AD=62330!
50 FOR I=0 TO 1
60 READ A:POKE AD+I,A
70 NEXT I
80 REM *** MACHINE DO ***
90 INPUT"INPUT 0-254";N
100 POKE 63911!,N:EXEC AD
110 PRINT"N+1=";PEEK(63911!)
120 PRINT" ｺﾉｱﾄ AD ｶﾗ BSAVE ｼﾃｵｹﾊﾞ"
130 PRINT" ﾋﾂﾖｳﾅﾄｷ BLOAD ﾃﾞﾖﾋﾞﾀﾞｾﾙ｡"
140 DATA 60,201
```

## 4.2 キャラクタ定義について

$N_{82}$-BASIC が扱えるキャラクタは全部で256種類あり，それぞれキャラクタコードの（第7章資料参照）0から255に対応しています．この中で初めから文字や記号，コントロール文字が割り当てられているのは0～130及び160～223でその他はユーザが自由に文字や記号を定義して使うことができます．この中でキーボードから入力できるものはキャラクタコード131～159の29種類でその他（224～255）は CHR$関数を用いて出力します．

### 4.2.1 キャラクタの構造

1つのキャラクタは必ず横が6ドット，縦が8ドットという構成になっています．キャラクタを定義するときはこれを縦1列8ドットが6組つながってできているものと考えてください．

### 4.2.2　キャラクタデータの作り方

4.2.1の例のようなデータは8ドット1組で1バイト，6組で6バイトのデータとなり，1つのキャラクタは6バイトのデータで表現されることがわかります．（1バイト＝8ビット＝$2^8$＝256種類の状態を表せる：データの単位）1つのデータは次のようにして得られる数値で表されます．ドットがある所を1，ない所を0とします．

例であげたキャラクタの残りの5つのデータも同様に計算するとそれぞれ，132，255，63，36，230となり，計6つのデータができあがります．

### 4.2.3　キャラクタ定義

4.2.2で作りあげた6バイトのデータをRAM上に書き込み，データの開始番地をワークエリアに書に込めば定義の作業は終わりです．書き込みはすべてPOKE文で行います．一歩間違えばRAM上のファイルなどが破壊されることもあるので，実行前に必要なファイルはカセットなどにセーブしておいた方が安心です．

まずデータを書いておくメモリ領域をCLEAR文で確保します．例えば60000番地から書き始めるなら，〈CLEAR 256, 60000〉を実行します．次に6バイトのデータを，順番に〈POKE 60000, 140〉，〈POKE 60001, 132〉……のように書き込んでいきます．6バイトのデータを書き終わると1キャラクタの終りです．更に別のキャラクタデータを続けて書き込んでいけば2キャラクタ目以降も同様に定義できます．(何も書き込

まなくても，もともとRAM上に値があればそれをデータと解釈するのでデタラメなキャラクタが定義されていることになります．）

最後に，データを格納した先頭番地をワークエリアに書き込めば定義は完了です．先頭番地は次のように計算して書き込みます．

60000（例の場合の先頭番地）を256で割ると，商234余り96

この商234を65216番地に書き込む．（〈POKE 65216, 234〉を実行する）

この余り96を65215番地に書き込む．（〈POKE 65215, 96〉を実行する）

その結果CHR$ (131)（及び GRPH ＋ V のキー入力）に60000番地から60005番地までのデータで構成されるキャラクタを定義したことになります．2キャラクタ目以降も60006番地からのメモリ内容をデータとして，順にCHR$ (132) 以降に定義されます．こうして一度定義した一連のキャラクタはデータを書き換えるか，65215番地及び65216番地の内容を新たに書き込まない限りいつまでも使えます．

注意：65216番地及び65215番地はキャラクタ定義専用のワークエリアで，常にこの番地のままです．データそのものを62336番地以降に書き込むと暴走の原因になります．気をつけてください．（データの先頭番地を61970にするとCHR$ (255) のデータの最後が62335番地になります．）

定義可能なキャラクタコードのうちキーボードから入力できるものは次のような対応になっています．（すべてグラフィックキーを押しながら対応キーを押します．）

CHR$（〈キャラクタコード〉）←→ GRPH ＋英字・記号キー

| コード | 対応キー | | | | | | |
|---|---|---|---|---|---|---|---|
| 131 | V | 138 | D | 146 | E | 154 | @ |
| 132 | B | 139 | F | 147 | R | 155 | ¥ |
| 133 | N | 140 | G | 148 | T | 156 | , |
| 134 | M | 141 | H | 149 | Y | 157 | . |
| 135 | L | 142 | J | 150 | U | 158 | ／ |
| 136 | A | 143 | K | 151 | I | 159 | ∧ |
| 137 | S | 144 | Q | 152 | O | | |
| | | 145 | W | 153 | P | | |

キャラクタコード160〜223はカナ文字で使用している．224〜255はCHR$を使ってのみ出力できる．

### 4.2.4 定義したキャラクタのセーブ

一度定義した一連のキャラクタは，そのデータをファイルとしてセーブしておけばまた後で使う事ができます．

キャラクタデータはメモリ内容そのものになるので，BSAVE 命令で〝.CO″ファイルとしてセーブします．(詳しくは第1章ファイル，第2章 BSAVE を参照) 例えばゲームに使うキャラクタ用のデータを10キャラクタ分 (10＊6＝60バイト)，60000番地からセットした場合なら，

BSAVE 〝GAME. CO″, 60000, 60

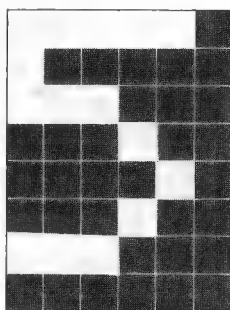
のようにします．(ファイル名〝GAME″はもちろん好きなように付けられます．)

これを使う時は

BLOAD 〝GAME. CO″

を実行すればよいだけです．

こうして様々なキャラクタセットをファイルとしてとっておけば，BLOAD によって多くのキャラクタを簡単に扱う事ができます．

この時注意するのは，データの開始番地を一度決めたらいつも同じにしておくことです．もしデータの開始番地が違う時は4.2.3でやったようにそのデータ開始番地をワークエリアに書き込まなければなりません．

# サンプルプログラミング

# 5 サンプルプログラミング

この章は，いくつかのプログラムリストとその簡単な解説から構成されています．BASICの命令や関数の一つ一つの使い方がわかっても，その組み合わせ方がわからなければ，なかなか役に立つプログラムは作れません．ここで紹介するプログラムは，それ自身で仕事ができるようになっていますが，興味のある方は解析してもっと使いやすくなるように改良したり，更に大きなプログラムの一部として使えるようにしてみてください．

## PSET ルーチン

PSET命令を使って線を引く，箱や円などを描くためのプログラムです．必要な部分だけをサブルーチンにして，自作のプログラムに応用してみてください．

```
10 '********* LINE BOX CIRCLE *********
20 SCREEN 0,0:CLS
30 PRINT
40 PRINT "  PSET ノ オウヨウ "
50 PRINT
60 PRINT "     1 LINE   "
70 PRINT "     2 BOX    "
80 PRINT "     3 CIRCLE"
90 PRINT
100 INPUT "        ドレ ヲ カキマスカ ";A$
110 ON VAL(A$) GOTO 130,260,400
120 BEEP:GOTO 20
130 '************** LINE **************
140 CLS:PRINT
150 INPUT "  シテン X ザヒョウ";X0:IF X0<0 OR X0>239 THEN BEEP:GOTO 150
170 INPUT "  シテン Y ザヒョウ";Y0:IF Y0<0 OR Y0>63  THEN BEEP:GOTO 170
190 INPUT "  シュウテン X ザヒョウ";X1:IF X1<0 OR X1>239 THEN BEEP:GOTO 190
210 INPUT "  シュウテン Y ザヒョウ";Y1:IF Y1<0 OR Y1>63 THEN BEEP:GOTO 210
230 CLS:GOSUB 520
240 FOR I=0 TO 1000:NEXT:BEEP:GOTO 20
260 '************** BOX ***************
270 CLS:PRINT
290 INPUT "          X ザヒョウ";X0:IF X0<0 OR X0>239 THEN BEEP:GOTO 290
310 INPUT "          Y ザヒョウ";Y0:IF Y0<0 OR Y0>63 THEN BEEP:GOTO 310
330 INPUT "          X ザヒョウ";X1:IF X1<0 OR X1>239 THEN BEEP:GOTO 330
350 INPUT "          Y ザヒョウ";Y1:IF Y1<0 OR Y1>63  THEN BEEP:GOTO 350
370 CLS:GOSUB 660
380 FOR I=0 TO 1000:NEXT:BEEP:GOTO 20
400 '************* CIRCLE *************
410 CLS:PRINT
420 PRINT "    チュウシン ノ ザヒョウ"
430 INPUT "           X ザヒョウ";X0:IF X0<0 OR X0>239 THEN BEEP:GOTO 430
450 INPUT "           Y ザヒョウ";Y0:IF Y0<0 OR Y0>63  THEN BEEP:GOTO 450
470 INPUT "              ハンケイ";R:IF R<0 THEN BEEP:GOTO 470
490 CLS:GOSUB 740
500 FOR I=0 TO 1000:NEXT:BEEP:GOTO 20
520 '************ SUB LINE ************
530 XD=ABS(X1-X0):YD=ABS(Y1-Y0)
540 XS=SGN(X1-X0):YS=SGN(Y1-Y0)
550 IF XD>YD THEN 600
560 F=-1:T=X0:X0=Y0:Y0=T
570 T=X1:X1=Y1:Y1=T
580 T=XD:XD=YD:YD=T
590 T=XS:XS=YS:YS=T
600 R=XD/2
610 IF F THEN PSET(Y0,X0) ELSE PSET(X0,Y0)
620 IF X0=X1 THEN RETURN
630 X0=X0+XS:R=R+YD
640 IF R>=XD THEN R=R-XD:Y0=Y0+YS
650 GOTO 610
660 '************ SUB BOX *************
670 FOR I=X0 TO X1 STEP SGN(X1-X0)
680 PSET(I,Y0):PSET(I,Y1)
690 NEXT
700 FOR I=Y0 TO Y1 STEP SGN(Y1-Y0)
710 PSET(X0,I):PSET(X1,I)
720 NEXT
730 RETURN
740 '*********** SUB CIRCLE ***********
750 FOR I=0 TO 1 STEP 1/(R*2)
760 II=I*I
770 X=R*I*2/(II+1)
780 Y=R*(1-II)/(II+1)
790 X2=X0-X:IF X2<0 THEN X2=0
800 Y2=Y0-Y:IF Y2<0 THEN Y2=0
810 X1=X0+X:Y1=Y0+Y
820 PSET(X1,Y1):PSET(X1,Y2)
830 PSET(X2,Y1):PSET(X2,Y2)
840 NEXT
850 RETURN
```

## キャラクタ定義プログラム

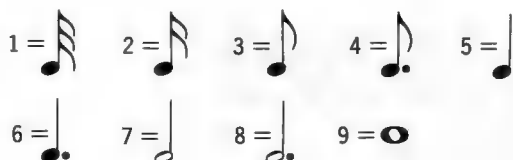
第4章で解説したキャラクタ定義という機能を用いると，非常に多くのキャラクタを自由に定義して使うことができます．しかし，そのデータを一つ一つ紙に書いて計算するのは大変です．このプログラムを使えば，スクリーンエディットの要領で一度に61個までのキャラクタが簡単に定義できます．また，一度作ったキャラクタセットはファイルとしてとっておけるので，何セットも作っておいてBLOADで次々とロードすれば何百，何千というキャラクタが扱えるようになります．ESCキーやEキーを使って定義をパスしたところは，以前定義したまま残されます（何も定義されてないとゴミが残っている事もあります）．

```
10 REM CHARACTER GENERATOR
20 REM USING ADRESS 61970-62335
30 CLEAR 256,61970!:DIM M(5,7):DEFINTB-Z
40 REM ***** INITIALIZE *****
50 SCREEN 0,0:CLS
60 POKE 65215!,18:POKE 65216!,242
70 H=131:C=0:AD=61970!
80 REM ***** MAIN LOOP1 *****
90 LOCATE 20,0:PRINT " ｼｮｳ ｷｰ"
100 LOCATE 15,1:PRINT "ｽﾍﾟｰｽ - ﾓｰﾄﾞ"
110 LOCATE 15,2:PRINT "ｶｰｿﾙ  - ｲﾄﾞｳ"
120 LOCATE 15,3:PRINT "´ESC´ - 1CHR ﾊﾟｽ"
130 LOCATE 15,4:PRINT "_ - 1ｷｬﾗｸﾀ ﾃｲｷﾞ"
140 LOCATE 15,5:PRINT "E - ｻｷﾞｮｳ ｵﾜﾘ"
150 LOCATE 10,7:PRINT "CHR$(";
160 PRINT MID$(STR$(H),2);")ｦ ﾃｲｷﾞﾁｭｳ｡";
170 X=0:Y=0:MX=0:MY=0:H=H+1:IF H=160 THEN H=224
180 FOR Y1=0 TO 63:PSET(36,Y1):NEXT:
190 REM ***** MAIN LOOP2 *****
200 IF T=0 THEN C$="ｹｽ" ELSE C$="ｶｸ"
210 LOCATE 10,0:PRINT C$
220 LOCATE X,Y:I$=INPUT$(1)
230 IF I$=CHR$(27) THEN 360
240 IF I$=CHR$(28) THEN X=X+1:IF X=6 THEN X=5 ELSE MX=MX+1
250 IF I$=CHR$(29) THEN X=X-1:IF X=-1 THEN X=0 ELSE MX=MX-1
260 IF I$=CHR$(30) THEN Y=Y-1:IF Y=-1 THEN Y=0 ELSE MY=MY-1
270 IF I$=CHR$(31) THEN Y=Y+1:IF Y=8 THEN Y=7 ELSE MY=MY+1
280 IF I$=CHR$(32) THEN T=NOT T
290 IF I$="E" OR I$="e" THEN 510
300 IF I$=CHR$(13) THEN GOSUB 400:GOTO 360
310 M(MX,MY)=-T:LOCATE X,Y
320 IF T THEN PRINT"#"; ELSE PRINT" ";
330 PSET(MX+40,MY+30,-T)
340 GOTO 200
350 REM ***** END OF LOOP*****
360 IF H=256 THEN 510
370 C=C+1:CLS
380 GOTO 90
390 REM ***** DATA POKE  *****
400 FOR X=0 TO 5
410 FOR Y=0 TO 7
420 M=M+M(X,Y)*2^Y
430 NEXT Y
440 POKE AD+C*6+X,M
450 M=0
460 NEXT X
470 FOR Q=0 TO 5:FOR R=0 TO 7:M(Q,R)=0
480 NEXT R,Q
490 RETURN
500 REM ***** LISTING *****
510 CLS:PRINT "ｲﾏ ﾃｲｷﾞｻﾚﾀ ｷｬﾗｸﾀ｡(131-159)"
520 FOR I=131 TO 159
530 PRINT CHR$(I);" ";:NEXT:PRINT
540 PRINT " ﾄ (224-255)"
550 FOR I=224 TO 255
560 PRINT CHR$(I);" ";:NEXT:PRINT
570 PRINT " ｶﾞ ｱﾘﾏｽ｡"
580 INPUT"BSAVE ｼﾏｽｶ(Y/N)";Y$
590 IF Y$="Y" OR Y$="y" THEN INPUT" ﾌｧｲﾙﾒｲ";N$ ELSE END
600 REM ***** FILE SAVE *****
610 BSAVE N$,61970!,366
620 END
```

## MUSIC プログラム

$N_{82}$-BASICのSOUND命令は，その音程を決める第一パラメータが音楽で言う半音よりずっと細かくなっているので，擬音などを作り出すには便利ですが，曲を演奏するとなると少しプログラムを工夫しなければなりません．このプログラムは音楽専用で，曲の入力，演奏の2つの部分にわかれています．入力はキーボードを一種の楽器の鍵盤に見たて，音の長さ（〝L″＋1～9初期設定は5），オクターブ（〝O″＋1～4初期設定は2），音程（ハ長調のド～シがキーボード上の〝Z″，〝X″，〝C″，〝V″，〝B″，〝N″，〝M″，半音は斜め上の〝S″，〝D″，〝G″，〝H″，〝J″に対応する）の順に入れていきます．音の長さは，次のような対応になっています．休符はスペースを入力します．

1 =　2 =　3 =　4 =　5 =

6 =　7 =　8 =　9 =

音の長さ，オクターブは変更しない時は省略する事ができ，自動的に前の値のままになります．入力時に ESC キーを押すと1音だけはやり直す事ができます．

音を20音位入力したところで E キーを押すと，そのパートが自動的に演奏されこれで良ければ次に進みます．だめだともう1度入れ直す事ができます．入力の時は画面に〝A″～〝G″（#は小文字）が表示され〝ラ″～〝ソ″（ハ長調）に対応しています．入力がすべて終わったら Q キーを押します．

入力が終わるとその曲をデータファイルにしておく事ができます．

データがファイルにある時や，入力終了後は演奏する事ができます．移調，変速の機能もあります．プログラムの指示に従ってください．

もっと長い曲を入れたい，入力，編集方法を変えたいという時は，実際のデータとして使っている文字型配列（原データ）と数値型配列（演奏用データ）の使い方に注目して改良してみてください．また，原データの構造がわかればTEXTモードで途中から直接書き変える事ができます．

```
10 REM *** MUSIC ***
20 CLEAR 2000,62336!:MAXFILES=1
30 DEFINT A-T:DEFSNG U-Y:DEFDBL Z
40 DIM A(48),M$(49).S(999),L(999)
50 SCREEN 0,0:Z=9394#
60 FOR I=0 TO 47
70 A(I)=Z:Z=Z/1.0594639#
80 NEXT I
90 FOR I=1 TO 9:READ LN(I):NEXT
100 DATA 4,8,16,24,32,48,64,96.128
```

```
110 REM *** MENU ***
120 CLS:PRINT" *** MUSIC ***
130 PRINT:PRINT" --- PLAY OR INPUT ---"
140 PRINT:INPUT"(P/I)";Y$
150 IF Y$="P" OR Y$="y" THEN 190
160 IF Y$="I" OR Y$="i" THEN 640
170 PRINT"????":BEEP:BEEP:GOTO 120
180 REM *** PLAY ***
190 CLS:PRINT" --- PLAYER ---"
200 PRINT:PRINT" ヨウイシテアル ファイルメイ"
210 INPUT" ヲ ニゥウリョク";N$
220 OPEN N$ FOR INPUT AS #1
230 S=0:E=0
240 IF EOF(1) THEN 270
250 LINEINPUT #1,M$(E)
260 E=E+1:GOTO 240
270 CLOSE:PRINT" ロート゛オワリ"
280 PRINT" テ゛ータ ヘンカン シマス。"
290 PRINT" O1G-O4F マテ゛ノ キョクハ イチョウ テ゛キマス。":PRINT" L1=4 トシテ ヘンソク テ゛キマス。"
300 INPUT" イチョウ シマスカ(Y/N)";I$
310 INPUT" ヘンソク シマスカ(Y/N)";Y$
320 IF I$<>"Y" AND I$<>"y" THEN 360
330 INPUT" ハンオン タンイ ノ イチョウ(-7 カラ +7)";D:IF D<-7 OR D>7 THEN 330
340 IF D>0 THEN FOR I=0 TO 41:A(I)=A(I+D):NEXT:GOTO 360
350 FOR I=47 TO 7 STEP -1:A(I)=A(I+D):NEXT
360 IF Y$="Y" OR Y$="y" THEN INPUT"V (.25 カラ 2)";V ELSE V=1
370 PRINT" ---スコシ オマチ クダ゛サイ---"
380 C=0:FOR I=0 TO E-1
390 T$=M$(I):GOSUB 540
400 NEXT I
410 BEEP:CLS
420 PRINT N$;"テ゛ータ ヘンカン オワリ。"
430 LOCATE 10,3:PRINT N$:LOCATE 10,4
440 PRINT" HIT ANY KEY !"
450 IF INKEY$<>"" THEN 450
460 IF INKEY$="" THEN 460
470 LOCATE 10,4:PRINT SPACE$(14)
480 FOR I=0 TO C-1:SOUND S(I),L(I)*V:NEXT I
490 INPUT"モウイチト゛(Y/N)";Y$
500 IF Y$="Y" OR Y$="y" THEN 430
510 IF I$="Y" OR I$="y" THEN PRINT" ショキ セッテイ ヲ ヤリナオシ マス。":RUN
520 GOTO 120
530 REM *** DATA COMPILER ***
540 FOR T=1 TO LEN(T$)
550 N=INSTR("CcDdEFfGgAaB LO",MID$(T$,T,1))
560 IF N>13 THEN GOSUB 600:GOTO 550
570 M=N+M:S(C)=A(M-1):L(C)=L:M=M-N
580 IF N=13 THEN S(C)=0
590 C=C+1:NEXT T:RETURN
600 IF N=15 THEN M=12*(VAL(MID$(T$,T+1,1))-1):T=T+2:RETURN
610 L=VAL(MID$(T$,T+1,1)):L=LN(L)
620 T=T+2:RETURN
630 REM *** INPUT ***
640 CLS:PRINT" --- INPUT ---"
650 S=0:E=0:C=0
660 INPUT" APPEND OR NEWDATA (A/N)";Y$
670 INPUT" ファイルメイ";N$
680 IF Y$="A" OR Y$="a" THEN OPEN N$ FOR APPEND AS #1 ELSE 710
690 PRINT" ツヅ゛キ ヲ":GOTO 730
700 REM *** NEW DATA ***
710 OPEN N$ FOR OUTPUT AS #1
720 PRINT"アタラシイ キョク ヲ"
730 PRINT"ニュウリョク シテクダ゛サイ。"
740 INPUT" ニュウリョク セツメイ(Y/N)";Y$
750 IF Y$="Y" OR Y$="y" THEN GOSUB 1260
760 CLS:L$="L5":O$="O2":S=C:M$(E)="":B=0:T$="":F=1:L=32
770 LOCATE 0,0:PRINT L$
780 LOCATE 3,0:PRINT O$
790 LOCATE 6,0:I$=INPUT$(1)
```

```
800 P=INSTR("ZSXDCVGBHNJM LOE"+CHR$(27)+"Q",I$)
810 IF P=0 THEN 790
820 I$=MID$("CcDdEFfGgAaB ",P,1)
830 IF F=1 THEN T$=L$+O$+I$
840 IF F=2 THEN T$=O$+I$
850 IF F=3 THEN T$=L$+I$
860 IF F=0 THEN T$=I$
870 IF B=0 THEN T$=L$+O$+I$
880 IF P=17 THEN IF F<>0 OR B=0 THEN 770 ELSE B=0:GOTO 1090
890 IF P=18 THEN IF S=C THEN E=E-1:GOTO 1120 ELSE 1120
900 IF P>13 THEN 960
910 X$=T$:B=1
920 PRINT I$;:M$(E)=M$(E)+T$
930 LOCATE 0,5:PRINT M$(E)+SPACE$(10);
940 GOSUB 540:SOUND S(C-1),L(C-1):F=0
950 GOTO 770
960 ON P-13 GOTO 970,1000,1030
970 IF S=C THEN F=1 ELSE IF F=2 THEN F=1 ELSE F=3
980 LOCATE 0,0:Y$=INPUT$(1):P=INSTR("123456789",Y$):IF P=0 THEN 970
990 L$="L"+Y$:GOTO 770
1000 LOCATE 3,0:Y$=INPUT$(1):P=INSTR("1234",Y$):IF P=0 THEN 1000
1010 IF S=C THEN F=1 ELSE IF F=3 THEN F=1 ELSE F=2
1020 O$="O"+Y$:GOTO 770
1030 LOCATE 0,3:PRINT "END OF PART";E;
1040 FOR I=S TO C-1:SOUND S(I),L(I):NEXT
1050 INPUT" OK(Y/N)";Y$:IF Y$="Y" THEN 1070
1060 C=S:PRINT"ﾔﾘﾅｵｼ":BEEP:GOTO 760
1070 S=C:IF E<49 THEN E=E+1:M$(E)="":F=1:B=0:CLS:GOTO 770
1080 BEEP:PRINT"OUT OF DATA SPACE":GOTO 1150
1090 M$(E)=LEFT$(M$(E),LEN(M$(E))-LEN(X$))
1100 C=C-1:BEEP:LOCATE 0,3:PRINT"1 STEP BACK":BEEP
1110 LOCATE 0,3:PRINT SPACE$(12);:GOTO 770
1120 PRINT:PRINT"END OF MUSIC"
1130 C=C+1
1140 REM *** END ***
1150 PRINT"YOUR MUSIC":FOR I=0 TO 200:NEXT
1160 FOR I=0 TO C-2:SOUND S(I),L(I):NEXT
1170 CLS:PRINT" ｾｰﾌﾞｶｲｼ"
1180 PRINT" ﾌｧｲﾙﾒｲ";N$:PRINT"HIT ANY KEY"
1190 IF INKEY$="" THEN 1190
1200 FOR I=0 TO E:PRINT #1,M$(I):NEXT I
1210 CLOSE:BEEP
1220 PRINT" ｾｰﾌﾞ ｵﾜﾘ｡ HIT ANY KEY"
1230 IF INKEY$="" THEN 1230
1240 GOTO 120
1250 REM *** EXPLAIN ***
1260 PRINT "    EXPLANATIONS"
1270 PRINT"1 CAPS ON ﾆ ｽﾙｺﾄ !"
1280 PRINT"2 ZSXDCVGBHNJM ﾉ ｷｰｶﾞｹﾝﾊﾞﾝ"
1290 PRINT"3 CcDdEFfGgAaB ﾆ ﾍﾝｶﾝ ｻﾚﾏｽ"
1300 PRINT"4 'E' ｦ ｵｽﾄ 1 ﾌﾞﾛｯｸ ｵﾜﾘ"
1310 PRINT"5 'Q' ｦ ｵｽﾄ ﾆｭｳﾘｮｸ ｵﾜﾘ"
1320 PRINT"6 'ESC' ｦ ｵｽﾄ 1 ｵﾝ ﾓﾄﾞﾙ"
1330 PRINT"7 SPACE ﾊ ｷｭｳﾌ"
1340 LOCATE 0,7:PRINT" HIT ANY KEY";
1350 IF INKEY$="" THEN 1350
1360 PRINT:PRINT"8 L=LENGTH(1-9),O=OCTAVE(1-4)"
1370 PRINT"9  20 ｵﾝ ｸﾞﾗｲ ｲﾚﾀﾗ 'E' ﾃﾞﾂｷﾞﾍ ｽｽﾑｺﾄ!"
1380 PRINT"10 PART 49 ﾏﾃﾞﾃﾞ ｵﾜﾘ"
1390 PRINT"11 L ﾔ O ﾊ 'ESC' ﾅｼﾃﾞﾅﾝﾄﾞﾃﾞﾓ ｶﾜﾘﾏｽ"
1400 LOCATE 0,7:PRINT" HIT ANY KEY";
1410 IF INKEY$="" THEN 1410
1420 RETURN
```

## DEMO プログラム

配列にデータを入れて，計算や表示に使う事がよくあります．特にこのデータとRND関数をうまく組み合わせると，面白い効果が作り出せます．本章のキャラクタ定義プログラムと組み合わせると，更に面白くなるかもしれません．

RUN したら，異なる（2種類の）文字や記号を入れてみてください．

```
10 REM ******** DEMO *********
20 CLEAR 256,62336!
30 SCREEN 0,0:CLS
40 DIM C%(39,7),X%(319,1):C=0
50 PRINT" READING DATA !"
60 FOR X=0 TO 39
70 FOR Y=0 TO 7
80 X%(Y*40+X,0)=X:X%(Y*40+X,1)=Y
90 READ C%(X,Y)
100 NEXT Y,X
110 REM ******* MAKE DATA ********
120 SCREEN 0,0:CLS:PRINT
130 PRINT" DATA SCRAMBLING !"
140 FOR I=0 TO 200
150 R=RND(1)*319
160 R1=RND(1)*319
170 N=X%(R,0):X%(R,0)=X%(R1,0):X%(R1,0)=N
180 N=X%(R,1):X%(R,1)=X%(R1,1):X%(R1,1)=N
190 NEXT
200 REM ******* PRINT *******
210 BEEP:CLS:PRINT CHR$(27)+"V"
220 PRINT" HIT ANY KEY ";:A$=INPUT$(1)
230 PRINT A$:PRINT
240 PRINT"  HIT ANOTHER KEY ";:B$=INPUT$(1)
250 PRINT B$:CLS
260 FOR N=0 TO 319
270 X=X%(N,0):Y=X%(N,1)
280 SOUND X*200+200,3
290 LOCATE X,Y
300 IF C%(X,Y)=1 THEN PRINT A$; ELSE PRINT B$;
310 NEXT
320 BEEP:LOCATE 0,0:PRINT CHR$(27)+"W"
330 FOR I=0 TO 500:NEXT
340 LOCATE 0,0:GOTO 130
350 DATA 0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0
360 DATA 0,1,0,0,0,1,0,0,0,0,1,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0
370 DATA 0,0,0,1,1,1,0,0,0,0,1,0,0,0,1,0,0,1,0,0,0,0,0,1,0,1,0,0,0,0,0,1
380 DATA 0,1,0,0,0,0,0,1,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0
390 DATA 0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,1,1,0,0,1,0,0,1,0,0,1
400 DATA 0,1,0,0,1,0,0,1,0,1,0,0,1,0,0,1,0,0,1,1,0,1,1,0,0,0,0,0,0,0,0,0
410 DATA 0,0,1,0,0,0,1,1,0,1,0,0,0,1,0,1,0,1,0,0,1,0,0,1,0,1,0,0,1,0,0,1
420 DATA 0,0,1,1,0,0,0,1,0,0,0,0,0,0,0,0,0,1,1,1,1,1,0,0,1,0,0,0,0,0,0,1
430 DATA 0,1,0,0,0,0,0,1,0,1,0,0,0,0,0,1,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0
440 DATA 0,1,0,0,0,0,0,1,0,1,1,1,1,1,1,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0
```

## GAME プログラム

ユーザーズマニュアルの第4章でも紹介しましたが，実際に面白いゲームを作るには様々な工夫が必要です．キャラクタ定義と組み合わせて作れば更に楽しいものになるでしょう．

カーソル移動キーでミサイル〝M″が移動し，スペースバーで発射です．1分間でゲームオーバーとなりますが，それ以上プレイしたい時は130行を変更してください．

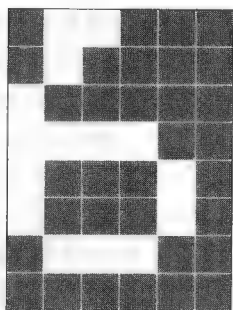
```
10 REM ***** GAME *****
20 DEFINT A-Z
30 SCREEN 0,0:CLS
40 TIME$="00:00:00"
50 SC=0
60 REM ***** START ****
70 X=RND(1)*35+1
80 LOCATE X,0:PRINT " >O< ";
90 I$=INKEY$
100 IF I$=CHR$(28) THEN M=M+1
110 IF I$=CHR$(29)THEN M=M-1
120 IF I$=" "THEN GOSUB 230
130 IF TIME$>"00:01:00" THEN 460
140 IF M<0 THEN M=37:LOCATE 0,6:PRINT"   ";
150 IF M>38 THEN M=1:LOCATE 38,6:PRINT"   ";
160 LOCATE M,6:PRINT " M ";
170 LOCATE 2,7:PRINT TIME$;
180 LOCATE 18,7:PRINT SC;"POINTS";
190 P=RND(1)*3-1:X=X+P
200 IF X<1 THEN X=1
210 IF X>35 THEN X=35
220 GOTO 80
230 REM **** MISSILE SUB ****
240 FOR Y=5 TO 0 STEP -1
250 LOCATE M+1,Y:PRINT "!";
260 SOUND Y*1000+1000,1
270 LOCATE M+1,Y:PRINT " ";
280 NEXT
290 IF M=X OR M=X+2 THEN SC=SC+1:BEEP:GOSUB 330:RETURN 70
300 IF M=X+1 THEN GOSUB 390
310 RETURN
320 REM *** ｺｱﾀﾘ ***
330 FOR I=0 TO 10
340 LOCATE X,0:PRINT" Ugu!";
350 FOR J=0 TO 20:NEXT:LOCATEX,0:PRINT "
360 SOUND 16000,1:NEXT
370 RETURN
380 REM *** ｵｵｱﾀﾘ ***
390 SC=SC+5:SOUND 440,10
400 FOR I=0 TO 10
410 LOCATE X-1,0:PRINT"Hieee!"
420 SOUND 1760,1
430 NEXT I
440 LOCATE X-1,0:PRINT"      "
450 RETURN
460 LOCATE 10,4:PRINT"End of GAME":END
```

## 成績管理プログラム

$N_{82}$-BASICで扱えるシーケンシャルファイル管理機能を利用して，成績管理（素点，平均，評準偏差）を行うプログラムです．項目や扱える件数などを改造すればいろいろなものに応用できます．

```
10 '************ ｾｲｾｷ ｼｮﾘ ***************
20 CLS:SCREEN 0,0
30 INPUT "ｶﾓｸｽｳ";NC
40 INPUT "ﾆﾝｽﾞｳ";NR
50 DIM D(NC,NR),NA$(NR),TIT$(NC),RSUM(NR),RMEAN(NR),SUM(NC),SSM(NC),MEAN(NC)
60 CLS:PRINT "ｶﾓｸﾒｲ"
70 FOR I=1 TO NC
80 LOCATE 0,2:PRINT SPACE$(40)
90 LOCATE 0,2:PRINT "ｶﾓｸﾒｲ";I;
100 INPUT TIT$(I)
110 NEXT:CLS
120 PRINT "ﾃﾞｰﾀ ﾆｭｳﾘｮｸ"
130 FOR J=1 TO NR
140 LOCATE 0,2:PRINT SPACE$(40):BEEP
150 LOCATE0,2:PRINT "No.";J;"ﾅﾏｴ";
160 INPUT NA$(J)
170 FOR I=1 TO NC
180 LOCATE 0,4:PRINT SPACE$(40)
190 LOCATE 0,4:PRINT TIT$(I);" ﾉ ﾃﾝｽｳ";
200 INPUT DA
210 D(I,J)=DA:RSUM(J)=RSUM(J)+DA
220 SUM(I)=SUM(I)+DA
230 SSM(I)=SSM(I)+DA^2
240 NEXT
250 LOCATE 0,4:PRINT SPACE$(40)
260 RMEAN(J)=RSUM(J)/NC
270 NEXT
280 FOR I=1 TO NC
290 MEAN(I)=SUM(I)/NR
300 SD(I)=SSM(I)/NR-MEAN(I)^2
310 NEXT
320 REM ************ ｼｭﾂﾘｮｸ *************
330 PRINT " ｶﾞﾒﾝ ｦ ﾄﾒﾙﾆﾊ SPACE ｷｰ ｦ ｵｼﾃｸﾀﾞｻｲ"
340 OPEN"LCD:"FOR OUTPUT AS #1
350 FOR I=0 TO 1000:NEXT:BEEP:CLS
360 TI=200:GOSUB 510
370 CLOSE:PRINT
380 INPUT "FILE ｦ ﾂｸﾘ ﾏｽｶ Y/N";Y$
390 IF Y$<>"Y" AND Y$<>"y" GOTO 450
400 INPUT"FILE ﾉ ﾅﾏｴ";A$
410 OPEN A$ FOR OUTPUT AS #1
420 TI=0:GOSUB 510
430 CLOSE
440 PRINT
450 INPUT "ﾌﾟﾘﾝﾄ ｼﾏｽｶ Y/N";Y$
460 IF Y$<>"Y" AND Y$<>"y" THEN END
470 OPEN "lpt:" FOR OUTPUT AS #1
480 TI=0:GOSUB 510
490 CLOSE:END
500 REM ******** ｼｭﾂﾘｮｸ ｻﾌﾞﾙｰﾁﾝ *******
510 PRINT#1,"        ";:FOR I=1 TO NC
520 PRINT#1,RIGHT$("     "+TIT$(I),5);
530 NEXT
540 PRINT#1," ｺﾞｳｹｲ";"   ﾍｲｷﾝ"
550 FOR J=1 TO NR
560 PRINT#1,LEFT$(NA$(J)+"        ",8);
570 FOR I=1 TO NC
```

```
580 PRINT#1,USING"#####";D(I,J);:NEXT
590 PRINT#1,USING"###### ####.#";RSUM(J);RMEAN(J)
600 IF TI<>0 THEN IF INKEY$=" " THEN A$=INPUT$(1)
610 FOR T=0 TO TI:NEXT:NEXT
620 PRINT#1,
630 PRINT#1," ゴウケイ  ";
640 FOR I=1 TO NC
650 PRINT#1,USING"#####";SUM(I);:NEXT
660 PRINT#1,
670 PRINT#1," ヘイキン   ";
680 FOR I=1 TO NC
690 PRINT#1,USING"#####";MEAN(I);:NEXT
700 PRINT#1,
710 PRINT#1," ヘンサ    ";
720 FOR I=1 TO NC
730 PRINT#1,USING"###.#";SQR(SD(I));:NEXT
740 PRINT#1,
750 RETURN
```

|  | エイゴ | スウガク | コクゴ | ゴウケイ | ヘイキン |
|---|---|---|---|---|---|
| アカシマ | 71 | 78 | 73 | 222 | 74.0 |
| イワセ | 53 | 78 | 80 | 211 | 70.3 |
| エンドウ | 83 | 62 | 48 | 193 | 64.3 |
| オオタ | 78 | 91 | 45 | 214 | 71.3 |
| カワナ | 73 | 46 | 43 | 162 | 54.0 |
| ナガシマ | 43 | 75 | 72 | 190 | 63.3 |
| ニシムラ | 80 | 71 | 72 | 223 | 74.3 |
| フルヤ | 78 | 64 | 69 | 211 | 70.3 |
| マツムラ | 68 | 82 | 70 | 220 | 73.3 |
| ヨシカワ | 60 | 58 | 93 | 211 | 70.3 |
| ゴウケイ | 687 | 705 | 665 |  |  |
| ヘイキン | 69 | 71 | 67 |  |  |
| ヘンサ | 12.3 | 12.5 | 15.4 |  |  |

# プログラミングの問題と対策

# 6 プログラミングの問題と対策

本章は BASIC のプログラミングを始めてまだ日の浅い方のために，特に設けられた章です．プログラムの中に潜む誤りのことをバグ（虫）と呼びますが，バグには大きくわけて 2 種類あります．一つはプログラムは走るが結果が予定と違う場合などで，もう一つはプログラムを走らせるとエラーを出してストップしてしまう場合です．それぞれについて問題と対策を考えてみます．

## 6.1 プログラムの実行結果がおかしい場合など

### ①以前正常に動いたプログラムが誤動作する

**原因** プログラムが変化している．知らずにプログラムをエディットしてしまった．

**解説** これは主に〝．BA″ファイルをロードして改良したとき起こりがちです．カセット上などのファイルと違って本体RAM上の〝．BA″ファイルはアクセス中（FILESを実行するとアスタリスク〝*″が示される）にエディットするとファイルそのものが変化してしまいます．新たに前のままのプログラムをロードし直したつもりでいても，変化した後のプログラムがロードされます．〝.DO″ファイルではこのようなことは起こりません（第1章1.17ファイル参照）．

いずれにせよ常にLISTやFILESを実行して，BASICがどういう状態にあるか知る習慣をつけておくことが大切です．

### ② STOPキーが効かず，制御できない

**原因** POKE文を間違って使い，PC-8201が暴走している．カセットを対象とした入出力命令を実行中である．

**解説** POKE文やEXEC文を良く理解せずに使うと，このようなことが起きます．いくら暴走しても別に機械に悪影響が出るようなことはないので実験してみるのはかまいませんが，RAM上に必要なファイルなどがある場合，それらは破壊されることもあります．

暴走してしまったら本体の電源スイッチを切り，再び入れればほとんどの場合メニュー画面に戻ります．このときメニュー画面が正常でも暴走の度合によってはファイルの内容などが破壊されている場合があります．BASICモードにして，動作やファイルの内容を調べて異常があれば，もうコールドスタートに頼るしかありません．［SHIFT］＋［CTRL］を押しながらリセットすればすべて正常に戻ります．（もちろん自分で作ったファイルはすべてキレイに消去されてしまいますが）

また，LLIST，CLOAD，CSAVE，SAVE〝CAS：″などの外部機器に対する入出力命令実行中は［STOP］キーだけでは実行の中止はできません．［SHIFT］＋［STOP］を押してください．

### ③画面表示が思い通りにならない

**原因** 「画面の最下位行に何かを出力するとスクロールしてしまう.」,「出力する字間が思い通りにならない.」など

**解説** PRINT 文で出力する内容の後にセミコロン〝;〞やカンマ〝,〞がないと, BASIC は自動的に改行を行って次の出力に備えます. 従って画面最下位行にセミコロンやカンマなしで出力すると画面全体がスクロールし, 思い通りにならない場合があります.

また最下位行のいちばん右に何かを出力した場合はセミコロンをつけても改行及びスクロールが行われます. スクロールを禁止するには[ESC]キーと大文字の〝V〞を入力し, 再びスクロールを許可するには[ESC]キーと大文字の〝W〞を入力してやります.

字間が自由にできない場合はまず第2章の PRINT, PRINTUSING, 第3章の TAB, SPACE$などを読んでみてください. 特に数値を出力する場合, 数値の前には符号をつけるための空白またはマイナス〝−〞記号, 後には次の出力と区別するための空白が出力される点に注意してください. これらの空白を取ってしまうためには, STR$関数や MID$関数を使って数値を数字文字列に変換して出力してやる必要があります.

### ④計算した値が予定通りにならない.

**原因** 「変数の型が合っていない. 誤差を考慮していない.」,「演算の優先順位を誤解している.」,「変数名が重複している.」などの場合があります.

**解説** 変数の型が合っていないと型変換が行われ, 予想外の計果になることがあります. プログラムの先頭で DEFINT 文などを使ったときは要注意です.

例)
```
10  DEFINT A−Z
 ⋮  (プログラムが続く)
100 FOR  I = 0  TO  2  STEP .5
110 NEXT
 ⋮
```

このようなプログラムは100行と110行が無限ループになってしまいます. プログラムの先頭で〝I〞も含めて変数は整数型として宣言したので, 100行の FOR 文中のカウンター変数 I はいつまでたっても 0 のままです. (0.5を加

えても切り捨てられる)
変数の型が合っていないのと関連して，誤差の問題があります．

```
例) 10  A = 0
    20  A = A + .1
    30  IF A = 1   THEN END
    40  GOTO 20
```

このプログラムは一見正しいように見えますが，実は無限ループになってしまいます．型宣言をしていない変数はすべて単精度実数型となりますが，0.1のような少数はコンピュータの内部表現（2進数）では正確に表現できないので僅かながら誤差をもって格納されます．従って例のプログラムではA＝1という条件はいつまでたっても満たされません．(例えばAに0.1を10回加えたものを代入し，A#＝Aを実行してみるとA#は1.00000011920929のようになっているのがわかります.）分岐条件文に整数型以外の変数を使うときは，こういう点に注意し，IF A>1 THEN のようにしてやらねばなりません．
演算順位を誤解すると当然予想外の計算結果が生じてきます．第1章1.13の演算の優先順位をよく読んでください．論理演算などを含む複雑な式は必要以上にカッコ〝(　)″で囲んでしまうのも間違いを少なくするには良い方法です．
変数名が知らないうちに重複してしまうのもよくあるミスで，特にサブルーチンに飛んでメインルーチンで使っている変数を変更してしまうことがあります．また変数名は頭の2文字で判断されているので，例えばPRICE，PREPのような長い変数名を使うとき二つが同じであることに気がつかないとミスの原因になります．

### ⑤ TEXT モードから BASIC に戻れない

**原因** テキストを編集した結果，BASICプログラムの形式を損なってしまった．

**解説** これはBASIC上のバグではありませんが，プログラムの編集を行うときに起ることがあるので解説しておきます．スクリーンエディットとEDIT文で入いるTEXTモードのエディットは，やり方が異なっています．スクリーンエディットでは変更入力に [↵] キー入力が必要ですがTEXTモードでは不要です．TEXTモードでの [↵] キー入力は改行を意味し，間違って使

うと内容のない行や，行番号のないステートメントができてプログラムの形式がくずれてしまいます．BASICに戻れなくなったら，上記のような箇所を探して修正します．詳しくは第1章1.16からのプログラム編集の項を参照してください．

## 6.2 エラーを出してプログラムが止まってしまう場合

BASICのプログラムでは，不可能な命令や，理解のできない命令に対してエラーコードを出力して実行を停止します．プログラムを作っていてあまりエラーばかり出ているとイヤになるかもしれませんが，もしBASICにエラートラップ機能がなかったらプログラムの間違いを見つけるのが非常に難かしくなってしまいます．

エラーの原因と解説は，見やすくするためにアルファベット順のエラーコードに従ってまとめておきます．コード番号はエラー別に決まっていて，そのエラーが起ったとき予約変数ERRに代入されます．N-BASIC，$N_{88}$-Disk BASICのエラーメッセージとの対応は第7章のエラーコード表を参照してください．

### ?AO Error コード番号53（Already Open）

**意味** 同じファイル番号で二度OPEN文を実行しようとした．
OPENしたままのファイルをKILLしようとした．

**原因** ● OPEN文の実行後，CLOSEするのを忘れている．
● KILLの前にCLOSEするのを忘れている．

**解説** いずれの場合も，OPENしたファイルはCLOSEすることを忘れないようにします．同時に複数のファイルをOPENするときは，MAXFILES文によって必要なファイル番号を確保しなければなりません．

### ?BN Error コード番号51（Bad file Number）

**意味** 使用できないファイル番号を使おうとした．

**原因** ● OPEN文で指定してないファイル番号をPRINT #文などで使っている．
● MAXFILES文で定めた最大のファイル番号より上のファイル番号をOPEN文で使っている．

**解説** PRINT #文などで使うファイル番号は必ず，それ以前にOPEN文でファイルを割り当ててあるものでなければなりません．ファイル番号は初期設定で

は1しか使えないので2-15を使うときは，MAXFILES文で指定しておかねばなりません。

## ?BO Error コード番号23（Buffer Overflow）

**意味** データが入力バッファからあふれた。

**原因** ●データが一度にバッファ容量以上に入ってきた。

**解説** $N_{82}$-BASICでは，自動的にあふれたデータを無視してしまうので通常はこのエラーは発生しません。

## ?BS Error コード番号9（Bad Subscript）

**意味** 配列の添字が0からDIM文によって宣言されたとき大きさの範囲を越えている。

**原因** ●添字に変数を使っている場合，この変数の値が演算の結果大きくなり過ぎている。

●配列の次元数を誤まっている。

**解説** エラーが起こったらすぐに，その添字として使っている変数の内容をPRINTして調べます。予定外に大きければ，その変数を使った演算が原因です。宣言していない配列は常にその添字の上限を10とします。宣言していない4次元以上の配列を使ったり，非常に大きな配列を宣言しようとしたときもこのエラーが発生します。

## ?CF Error コード番号58（Closed File）

**意味** まだOPENしていないファイルをアクセスした。

**原因** ●まだOPEN文でファイルを割り当てていないファイル番号を使って，PRINT #などを実行した。

**解説** PRINT #，INPUT #，PRINT # USING，LINEINPUT #文及びINPUT$関数で使うファイル番号は，事前にOPEN文によって対応するファイルを割り当てておかなければなりません。

## ?CN Error コード番号17（Continue Not possible）

**意味** プログラムの続行ができない。

**原因** ●プログラムをストップさせてから編集してしまうと，CONT 命令によるプログラムの再開，続行ができなくなる.

●プログラム中にステートメントとして CONT が書いてある.

**解説** プログラムをストップさせて変数の値を調べたり，変数に値を代入したり LIST を取ったりしても CONT 文による実行の再開は可能ですが，プログラムを一部でも書き換えたり，CLEAR 文を実行したり，画面に LIST したプログラム上にカーソルを移動して [↵] を押しただけでもこのエラーの原因になります.

## ? DD Error コード番号10（Duplicate Definition）

**意味** 同じ名前の配列を２度宣言した.

**原因** ●一度宣言した配列を再定義することはできない．宣言しないで使った配列も，添字の上限を10として自動的に宣言したことになるので再定義はできない.

**解説** 一度宣言した配列は，NEW，RUN，CLEAR などの命令が実行されるまでは再定義できません．これらの命令が実行されると他の変数もすべてクリアされてしまいます．特に配列の添字の上限だけを変えることはできないので，こういう場合は別の名前の配列を新しく宣言して定義するしかありません.

## ? DS Error コード番号56（Direct Statement in file）

**意味** アスキー形式ファイルをロード中プログラム以外のものがあった.

**原因** ● LOAD 命令によってアスキーセーブされたプログラムをロードするとき，ファイルの中に行番号のないステートメントがあるとプログラムとしてロードできなくなってしまう.

**解説** LOAD 命令はプログラムをロードするためのコマンドですから，TEXT モードで編集，作成した BASIC のプログラム形式以外のテキストファイル（".DO" ファイル）をロードしようとするとこのエラーが起こります．プログラムを作成，セーブするときに BASIC モードで行っていればこのようなことにはなりません.

## ? DU Error コード番号25 (Device Unavailable)

**意味** 指定したデバイスが使用できる状態にない.

**原因** ●指定したデバイスに何らかの異常がある.

**解説** 指定したデバイスが接続してない時などは通常は“? FC Error”となり，このエラーは起きません.

## ? EF Error コード番号54 (End of File)

**意味** ファイルのデータを読み尽した後にINPUT # 文が実行された.

**原因** ● INPUT #, INPUT$ (〈文字数〉, 〈ファイル番号〉), LINEINPUT #を実行する回数がファイルのデータ数より多い.

**解説** INPUT # などでデータを次々に読み出すとき，ファイル中のデータ数以上読ませないようにします. ファイル中のデータ数がわからないときは，EOF関数を使って現在読んだデータが最後かどうかを確めながらINPUT # 文を実行していきます.

## ? FC Error コード番号 5 (illegal Fuction Call)

**意味** 関数やステートメントの呼び方が間違っている.

**原因** ●パラメータの値がおかしい.

●値が 0 から255の間になければならないもの
CLOSE, COLOR, ERROR, LOCATE, MOTOR, ON GOTO や ON GOSUB の飛び先を決める式の値, OUT, POKE で書き込むデータ, POWER, PRESET, SCREEN, SOUND の第 2 パラメータ, CHR$, EOF, INP, INPUT$, INSTR, LEFT$, MID$, RIGHT$, SPACE$, STRING$, TAB.

●値が正または 0 でなければいけないもの
CLEAR の第 1 パラメータ，配列の添字, SQR, LOG ( 0 を除く)

●その他の値の範囲をもつもの
KEY, MAXFILES, SOUND の第 1 パラメータ, WIDTH, ファイル番号 ( 0 から255だが BN Error のときもある), LINE 及び PSET (LCD上では 0 ～255の値とするが CRT を接続すると違う).

● PRINT USING での桁指定が24桁を越えている.

●アクセス中のファイルを KILL しようとした.

● NAMEで既にある名前を新しい名前として他のファイルにつけようとした.

● 〝. BA″ファイルをMERGEしようとした.

● RENUMで行の順番を入れ換えようとした.

●接続されてないデバイスを使おうとした.(SCREEN 1など)

**解説** 原因が多岐にわたり,頻発するエラーの1つです.各関数やステートメントの使い方を区別して覚えることが大切です.プログラム中のエラーが生じるのはほとんどの場合,パラメータに使用している変数が演算の結果予定外の値になってしまうことによります.これはパラメータが出た直後にその変数の値をPRINTしてみればすぐわかります.パラメータに使う変数名は極力わかり易いものにして,他と区別するようにすると良いでしょう.

## ?FF Error コード番号52 (File not Found)

**意味** 指定したファイルが見つからない.

**原因** ● LOAD, KILL, OPENなどの命令で使う<ファイル名>が指定したデバイス上にない.(〝CAS:″ではいつまでも探し続けます.)

**解説** <ファイル名>はわかり易い名前をつけ,拡張子も意識しておくこと.またBLOAD命令で〝.CO″以外の拡張子をもつファイルをロードしようとしても,やはり〝.CO″として扱われるため,このエラーになることがあります.

## ?FL Error コード番号57 (Filing Limit)

**意味** もう新しいファイルが作れない.

**原因** ●ファイルが多すぎて新しいファイルを登録するエリアがない.

**解説** デバイスによって異なりますが,本体RAM上ではバンク1あたり21個までしかファイルを作ることはできません.メニュー画面に戻ってファイル名を登録する場所がなければ,もう一杯であることがわかります.不要なファイルをKILLして場所をあけるか,他のデバイスにファイルを移す必要があります.

## ?ID Error コード番号12 (Illegal Direct)

**意味** ダイレクトモードで使用できないステートメントを使った.

**原因** ●コマンドとして使えないステートメントをコマンドレベル(ダイレクト

モード）で使った.

**解説** $N_{82}$-BASICではこのようなステートメントは存在しないので，正常に動作している限りこのエラーは発生しません．N-BASICのダイレクトモードでは使えないINPUTも$N_{82}$-BASICではIDエラーにはなりません．

## ? IE Error コード番号50（Internal Error）

**意味** BASIC内部でエラーが発生した.

**原因** ● BASICインタープリタ内に何らかの異常がある.

**解説** 内部のエラーなのでBASICが正常である限り発生しないエラーです.

## ? IO Error コード番号24（I/O error）

**意味** 周辺装置との入力出力上のエラー

**原因** ●周辺装置との入出力中に強制的にストップをかけた.

●周辺装置との入出力がうまくいかない.

**解説** カセットテープを使っている場合，傷などがないか，レコーダのヘッドは汚れていないか，ボリュームやテープの回転はスムーズになっているかなどに気をつけます．強制的にストップ（SHIFT＋STOP）した場合は，単にエラーメッセージが出力されるだけと考えてかまいません．

## ? LS Error コード番号15（Long String）

**意味** ストリング（文字列）が長すぎる.

**原因** ●文字列演算の結果，文字列の長さが255を越えてしまった.

**解説** 文字列の長さは255までと決まっているので，A$＝A$＋"LONG"などの演算を操り返しているとA$の長さが256以上になった時点でエラーが発生します．文字列演算が無限ループの中にあったり，演算の要素になる文字列（特に文字型変数）が予定外の長さになっているということのないように気をつけます．どうしても256以上の長さのストリングを扱いたければ，2つ以上の変数名を使って分割処理しなければなりません．

## ? MO Error コード番号22（Missing Operand）

**意味** 必要なオペランドが欠けている.

原因　●コマンドや関数で省略できないパラメータや引数が欠けている．
●代入文の右辺がない．
●演算子があるのに被演算子がない．

解説　コマンド・ステートメント，関数の使い方や第1章の式と演算などをもう一度確認してください．

### ? NF Error コード番号1 (Next without For)

意味　FOR文がないのにNEXT文が出てきた．

原因　● FORの数がNEXTの数と合っていない．
● FOR～NEXTループの中にGOTOやGOSUBで飛び込んでいる．
● FOR～NEXTの多重ループが正しい入れ子になっていない．

解説　慣れないうちは，FORの数とNEXTの数は必ず一致するようにプログラムします．入れ子構造を理解するためには，NEXTの後の制御変数(カウンタ)も省略したり，複数をまとめて指定しない方が見やすくなります．IF文のTHEN以下にNEXTを置くような変則的なことも，初めのうちは避けた方がよいでしょう．

### ? NM Error コード番号55 (file Name Mismatch)

意味　不適当なファイル名を使った．

原因　● LOAD, SAVE, KILL, NAME, OPEN, RUNなどの命令でファイル名を指定するとき，ファイル名に使えない文字や記号があったり，文字数が7文字以上になっている．
● BSAVE, BLOAD以外のコマンドで〝. CO″ファイルを扱おうとした．
● OPEN文で〝. DO″以外のファイルを指定した．

解説　ファイル名の制限をもう一度確認してください．コマンドによって扱えるファイルの種類が決まっているので，拡張子〝. BA″，〝. DO″，〝. CO″の意味も理解しておいてください．

### ? NR Error コード番号19 (No Resume)

意味　RESUME文がない．

原因　●エラー処理ルーチンにRESUME文がない．

解説　エラー処理ルーチンは必ずRESUME, END, ON ERROR GOROのどれ

かで終わっていなければなりません.

## ?OD Error コード番号4 (Out of Data)

**意味** READ文によって読まれるべきデータが足りない.

**原因** ● DATA文のデータ数が足りない.

● RESTORE文の使い方が間違っている.

**解説** 同じデータを何度も読ませる場合や, データをグループごとに違った行にまとめておいて扱うときはRESTORE文を使いますが, RESTORE文で指定した行番号が間違っているとこのようなエラーが発生します. READ文とDATA文の対応を正しく行ってください.

## ?OM Error コード番号7 (Out of Memory)

**意味** メモリが足りない.

**原因** ●プログラムが長すぎてメモリの中に収まりきらない.

●プログラムは収まっても, それを走らせるのに必要なメモリ(変数, スタック用など) が足りない.

●配列が大きすぎて, メモリ上にその領域をとることができない.

●フリーメモリ以上のストリング領域をとろうとした.

● FOR文やGOSUB文によるネスティング (入れ子) が深くなりすぎて, スタックが一杯になった.

●既にファイルが大きなメモリ領域を占めていて, 新しくファイルを作ったりプログラムを走らせるメモリ領域が足りない.

●機械語用のメモリ領域を(既にプログラムやファイルで使っている領域を壊す程) 大きくとろうとした.

**解説** ファイルをセーブしてある領域が大きい時は, KILL命令でファイルを削除します (必要ならカセットなどに移しておく). それ以外の場合はとにかくFRE関数を使ってフリーメモリを調べてみます. プログラムも短かく, ファイルをセーブしてある領域も小さいはずなのに, フリーメモリが異常に少ないときは, MAXFILES文で多くのバッファをとったり, CLEAR文でストリング領域を大きくとったり, 機械語領域を指定したまま忘れている可能性が大です. CLEAR文を使って初期設定し直してもまだだめなら, 変数や配列をできるだけ整数型にしてみます. それでもだめならメモリを増設するしかありません. (増設にも限界はあります.)

また，間違って10 GOSUB 10のように無限にサブルーチンコールを繰り返してしまうと，戻り番地を覚えるためのスタックが無限にふくれ上り，メモリが足りなくなってしまいます．

## ？OS Error コード番号14（Out of String space）

**意味** ストリング領域が足りなくなった．

**原因** ●ストリング（文字列）を扱うための十分な領域が確保されていない．

**解説** 例えばCLEAR 256（初期設定値）を実行すると文字型変数はその長さが合計256まで使えることになります．これでA$に128文字，B$に64文字，C$に64文字のように代入して使うことができるわけですが，これらの間で文字列演算を行う場合は更にその演算用の領域も確保しなければエラーになってしまいます．

## ？OV Error コード番号6（OVerflow）

**意味** 入力された数値，代入される数値，演算結果などが許される範囲を越えている．

**原因** ●整数演算の結果や整数型変数に代入する値が－32768から32767の範囲外である．

●実数演算などの結果が－1.70141E＋38から1.70141E＋38の範囲にない．

● PEEK, POKE, OUT, DIM などで指定するパラメータや添字の値が正しい範囲にない．

**解説** 変数の型とその範囲，コマンド・ステートメントのパラメータの範囲，関数で扱える範囲，論理演算の範囲などを把握しておくこと．

## ？PC Error コード番号59（PC-8001）

**意味** PC-8001の命令を使っている．

**原因** ● PC-8001の命令で実行できないものがある．

**解説** N-BASIC で書かれたプログラムを直接ロードした場合，$N_{82}$-BASIC で実行できないものがあると List をとった時＊ PC ＊と表示されています．これを書き換えて $N_{82}$-BASIC のプログラムに直します．通常は＊ PC ＊の部分は？SN Error や？FC Error として見つかるので，このエラーはまず起こりません．

## ? RG Error コード番号 3 (Return without Gosub)

**意味** GOSUB でサブルーチンに飛んだわけではないのに RETURN 文に出会った.

**原因**
- サブルーチンへ GOTO 文で飛び込んでいる.
- メインルーチン終了後に END 文がなく, 後に続いているサブルーチンの実行が始まってしまう.

**解説** メインルーチンの終りには END 文をおくようにする. また, サブルーチンとサブルーチンの間には必ず RETURN 文をおくようにすること.

## ? RW Error コード番号20 (Resume Without error)

**意味** エラーがないのに RESUME しようとした.

**原因**
- GOTO や GOSUB でエラー処理ルーチンの中へ分岐している.
- メインルーチン終了後に END 文がなく, 後に続いているエラ処理ルーチンの実行が始まってしまう.

**解説** "? RG Error" とよく似ています. メインルーチンの終りには END 文をおき, エラー処理ルーチンに不要意に飛び込むのを避けることです.

## ? SN Error コード番号 2 (Sy Ntax error)

**意味** プログラムの構文 (文の書き方) が間違っている.

**原因**
- タイプミスで BASIC 文法に合わないものがプログラムにまぎれ込んでいる.
- 関数や数式が代入文の左辺にあったり, ステートメントのように単独で使われている.
- 変数名が英字で始まっていない, 予約語を含んでいる場合など.
- マルチステートメントの区切り記号 ":" が抜けている.
- 行番号が 0 から65529の範囲にない.
- 行番号の指定に変数を使おうとしている.
- IF 文中で対応する THEN のない ELSE が使われている.
- コマンドのパラメータや関数の引数の個数に過不足がある.
- スクリーンエディット中に二つの行がつながってしまっている.

**解説** エラーがでたときに LIST. または [f・9] を実行すると, ほとんどの場合エラーのある行を示してくれるので, この中で原因となる間違いを探します.

単純なタイプミスなどはすぐわかりますが，次のようなものはなかなか気がつかないことがあります．

- 1とI，0とO，ピリオド〝.〟とコンマ〝,〟，コロン〝:〟とセミコロン〝;〟などの違い．
- 予約語(キーワード)を含む変数名を使おうとしている．COST，NOKORI，SHIFT，TANKなど
- 複雑な数式でカッコが正しく対応していない．
- 2行がくっついている(特に前の行が40文字ちょうどのとき[↵]キー入力を忘れることがある)．TEXTモードでエデットしてみるとよくわかる．([↵]キー入力が改行記号〝↵〟として表示される．)

## ?ST Error コード番号16（String formula Too complex）

**意味**　文字式が複雑すぎる．

**原因**　●文字式があまりに複雑で解析できない．

**解説**　めったに起こるエラーではありませんが，複雑すぎる文字式は2つ以上に分割しなければなりません．

## ?TM Error コード番号13（Type Mismatch）

**意味**　変数の型が合わない．

**原因**　●文字型変数に数値を代入しようとした．
●数値変数に文字列を代入しようとした．
●FOR文の制御変数に倍精度実数型を使おうとした．

**解説**　CHR$とASC，STR$とVALのような関数の使い方を確認してください．変数や定数について正しく理解しておくと間違いを見つけるのが容易になります．
また，FOR文の制御変数は整数型と単精度実数型しか使えません．

## ?UE Error コード番号21他（Unprintable Error）

**意味**　メッセージの定義されていないエラーを出そうとした．

**原因**　●ERROR文により定義されていないエラーコード番号のエラーを発生した．

**解説**　〝?UE Error〟はBASICの拡張やユーザ定義のために残されたコード番号

に割り当ててあるもので，21の他26～49及び60～255のコード番号に対応しています．

ERROR文でユーザが定義したエラーメッセージを出すには，エラー処理ルーチンでその処理を行わなければなりません．

### ?UF Error コード番号18（Undefined user Function）

**意味** 未定義ユーザ関数が呼ばれた．

**原因** ●未定義ユーザ関数が呼ばれた．

**解説** $N_{82}$-BASICではユーザ関数定義ができないので，通常このエラーは発生しません．

### ?UL Error コード番号8（Undefined Linenumber）

**意味** 行番号だけを入力した．指定された行番号が存在しない．

**原因** ● RENUM実行時に，参照すべき行番号がない．
● GOTO，GOSUBなどの分岐先の行番号が存在しない．
● RESTORE，RUNで指定した行番号が存在しない．

**解説** GOTO文の分岐先の行をプログラム編集時に消去したまま忘れていることがあります．編集で一行消してしまうときは，そこに分岐する命令がプログラム中にないことを確かめてください．

### ?／0 Error コード番号11（division by zero）

**意味** 0による除算を実行しようとした．

**原因** ●未定義の変数（初期値として0になっている）で除算を行った．
●演算の結果除数となる変数が0になっている．
● TAN関数の引数が$\pi$／2になっている．
●0に対して負のべき乗を行った．

**解説** エラーが出たときに，除数に使っている変数をPRINTして値を確かめます．0であればなぜ0になるか，プログラムの中でその変数を使って演算を行っている部分を検討してみます．除算には実数除算〝／〟，整数除算〝￥〟，剰余〝MOD〟の3つがあります．

## 6.3 Programming Hints

初心者の方は特に〝デバッグのために〟を参考にしてください.

デバッグのために.

① フローチャート(プログラムの流れ図)をしっかり書く.(特に長いプログラムで行きあたりばったりはバグのもと)

② マニュアルをしっかり読み,コマンドや関数は動作を確認して理解しておく.

③ 変数表を作り,変数名の重複を避ける.

④ デバッグ段階では特にできるだけ見やすいプログラムにすることを重視し,REM 文を使ってプログラムを区切り,マルチステートメントは避ける.

⑤ ある行が怪しいときは消してしまわず,まず REM 文にしてみる.

⑥ 必要な所に STOP 文を入れておき,変数の値の変化を確かめながら実行してみる.(CONT 命令を使う)

スピードアップのために.

① プログラム中のスペースや REM 文を取り去る.

② できるだけ整数型変数を使う.

③ NEXT 文中の制御変数指定が省略できれば,省略する.

④ できるだけマルチステートメントを使う.

⑤ 使用頻度の高い変数は,プログラムの始めの方で定義しておく(A=0のように使っておくということ).

⑥ よく使うサブルーチンはプログラムの先頭の方に書いておく.

⑦ 文字列用の領域は充分大きくとっておく.

⑧ 文字型変数はできるだけ使わない.

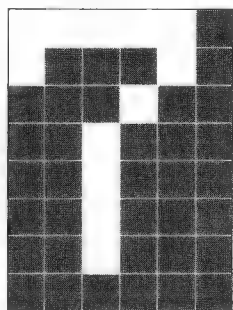
⑨ よく使うループの中の仕事を簡略化できないか見当する.このとき特に①,②,③,④を考える.

メモリ節約のために

① できるだけマルチステートメントを使う.

② プログラム中のスペースや REM 文を取る.

③ プログラム中に何度も現われるような定数は一度変数に代入して変数名で参照して使う.

④ 新しい変数を定義して用いるよりも使用していない古い変数があったらそれを使う．
⑤ 同じような仕事が多いときはそれをなんとか一つのサブルーチンにすることを考える．
⑥ 配列変数は必ず宣言して使う．自動宣言にすると例え不要でも添字が10まで使えるようにメモリを使ってしまう．
⑦ できるだけ整数型変数を使う．
⑧ ストリング領域は必要最小限しか確保しない．

# 資料

## ■予約語表

| | | | |
|---|---|---|---|
| AND | ELSE | LINE | READ |
| ABS | END | LOAD | RUN |
| ATN | EDIT | LIST | RESTORE |
| ASC | ERROR | LFILES | REM |
| BSAVE | ERL | LOG | RESUME |
| BLOAD | ERR | LOC | RIGHT$ |
| BEEP | EXEC | LEN | RND |
| COLOR | EXP | LEFT$ | RENUM |
| CLOSE | EOF | LOF | SCREEN |
| CONT | EQV | MOTOR | STOP |
| CLEAR | FORMAT | MERGE | STATUS |
| CLOAD | FOR | MOD | SAVE |
| CSAVE | FILES | MID$ | STEP |
| CSRLIN | FRE | MAX | SGN |
| CINT | FIX | MENU | SQR |
| CSNG | GOTO | NEXT | SIN |
| CDBL | GO TO | NAME | STR$ |
| COS | GOSUB | NEW | STRING$ |
| CHR$ | INPUT | NOT | SPACE$ |
| COM | IF | OPEN | SOUND |
| CLS | INSTR | OUT | THEN |
| CMD | INT | ON | TAB( |
| DATA | INP | OR | TO |
| DIM | IMP | OFF | TIME$ |
| DEFSTR | INKEY$ | PRINT | TAN |
| DEFINT | KILL | POKE | USING |
| DEFSNG | KEY | POS | VAL |
| DEFDBL | LOCATE | PEEK | WIDTH |
| DSKO$ | LPRINT | PSET | XOR |
| DSKI$ | LLIST | PRESET | |
| DSKF | LPOS | POWER | |
| DATE$ | LET | RETURN | |

■エラーコード表

| エラーメッセージ | コード | N-BASIC,N88-Disk BASICのメッセージ<br>(本書第6章で用いたエラー名称) | 意　味 |
|---|---|---|---|
| ? AO Error | 53 | File already open<br>(Already Open) | 同じファイルを二度OPENした。 |
| ? AT Error<br>(Disk) | 63 | Bad allocation table<br>(なし) | ディスクのFATが壊れている。 |
| ? BM Error<br>(Disk) | 60 | Bad file name<br>(なし) | ファイル名が適当でない。 |
| ? BN Error | 51 | Bad file number<br>(Bad file Number) | ファイル番号が適当でない。 |
| ? BO Error | 23 | Communications Buffer overflow<br>(Buffer Overflow) | 入力バッファが一杯になった。 |
| ? BS Error | 9 | Subscript out of range<br>(Bad Subscript) | 配列の添字が適当でない。 |
| ? CF Error | 58 | File not open<br>(Closed File) | ファイルがまだOPENされていない。 |
| ? CN Error | 17 | Can't Continue<br>(Continue Not possible) | CONTでプログラム実行再開ができない。 |
| ? DD Error | 10 | Duplicate Definition<br>(Duplicate Definition) | 同じ配列を二度宣言した。 |
| ? DF Error<br>(Disk) | 62 | Disk full<br>(なし) | ディスクが一杯で書き込めない。 |
| ? DN Error<br>(Disk) | 64 | Bad drive number<br>(なし) | ドライブ指定が適当でない。 |

| エラーメッセージ | コード | N-BASIC,N88-Disk BASICのメッセージ<br>(本書第6章で用いたエラー名称) | 意　味 |
|---|---|---|---|
| ? DS Error | 56 | Direct Statement in file<br>(Direct Statement in file) | アスキー形式のファイルが読み込めない。 |
| ? DU Error | 25 | なし<br>(Device Unavailable) | 指定デバイスが使用できない。 |
| ? EF Error | 54 | Input past end<br>(End of File) | ファイル中のデータを読み尽した。 |
| ? FC Error | 5 | Illegal function call<br>(illegal Function Call) | 命令や関数の使い方が適当でない。 |
| ? FE Error (Disk) | 61 | File already exists<br>(なし) | NAMEで既存のファイル名を使った。 |
| ? FF Error | 52 | File not Found<br>(File not Found) | 指定したファイル名が見つからない。 |
| ? FL Error | 57 | なし<br>(Filing Limit) | ファイルが多すぎる。 |
| ? FW Error (Disk) | 67 | File write protected<br>(なし) | ファイルに書き込み禁止属性が付いている。 |
| ? ID Error | 12 | Illegal direct<br>(Illegal Direct) | ダイレクトモードで使えない命令を使った。 |
| ? IE Error | 50 | Internal error<br>(Internal Error) | BASIC内部にエラーが生じた。 |
| ? IO Error | 24 | I/O error<br>(I/O error) | 入出力中にエラーが生じた。 |

## ■エラーコード表

| エラーメッセージ | コード | N-BASIC,N88-Disk BASICのメッセージ (本書第6章で用いたエラー名称) | 意　味 |
|---|---|---|---|
| ? LS Error | 15 | String too long<br>(Long String) | 文字型変数の内容が255文字を越えた。 |
| ? MO Error | 22 | Missing operand<br>(Missing Operand) | 必要なパラメータが欠けている。 |
| ? NF Error | 1 | NEXT without FOR<br>(Next without For) | FOR文がないのにNEXT文がある。 |
| ? NM Error | 55 | Bad file name<br>(file Name Mismatch) | ファイル名が適当でない。 |
| ? NR Error | 19 | No RESUME<br>(No Resume) | エラー処理ルーチンにRESUME文がない。 |
| ? OD Error | 4 | Out of data<br>(Out of Data) | 読むべきデータが足りない。 |
| ? OM Error | 7 | Out of memory<br>(Out of Memory) | メモリが足りない。 |
| ? OS Error | 14 | Out of string space<br>(Out of String space) | 文字列格納用のメモリ領域が足りない。 |
| ? OV Error | 6 | Overflow<br>(OVerflow) | 数値が大きすぎる。 |
| ? PC Error | 59 | なし<br>(PC-8001) | PC-8001の命令を使っている。 |
| ? RD Error (Disk) | 66 | Rename across disks<br>(なし) | 異なるドライブ間でNAMEを実行した。 |

| エラーメッセージ | コード | N-BASIC,N88-Disk BASICのメッセージ<br>(本書第6章で用いたエラー名称) | 意味 |
| --- | --- | --- | --- |
| ? RG Error | 3 | RETURN without GOSUB<br>(Return without Gosub) | GOSUB文がないのにRETURN文がある。 |
| ? RW Error | 20 | RESUME without error<br>(Resume Without error) | エラーがないのにRESUME文がある。 |
| ? SN Error | 2 | Syntax error<br>(SyNtax error) | 文の記述が間違っている。 |
| ? ST Error | 16 | String formula too complex<br>(String formula Too complex) | 文字式が複雑すぎる。 |
| ? TM Error | 13 | Type mismatch<br>(Type Mismatch) | 変数や定数の型が合わない。 |
| ? TS Error | 65 | Bad track/sector<br>(なし) | トラック，セクター番号の指定が適当でない。 |
| ? UE Error | 21 | Unprintable error<br>(Unprintable Error) | メッセージの定義されていないエラー。 |
| ? UF Error | 18 | Undefined user function<br>(Undefined user Function) | 未定義のユーザ関数を呼んだ。 |
| ? UL Error | 8 | Undefined line number<br>(Undefined Line number) | 指定した行が定義してない。 |
| ? /0 Error | 11 | Division by Zero<br>(division by zero) | 0による徐算を行った。 |
| | | | |

## ■コントロールコード表

| コード | 機　　能 | 対応キー |
|---|---|---|
| 0 | ヌルストリング | なし |
| 1 | | CTRL + A |
| 2 | | CTRL + B |
| 3 | 実行停止 | CTRL + C , STOP |
| 4 | | CTRL + D |
| 5 | カーソル以下を削除 | CTRL + E |
| 6 | | CTRL + F |
| 7 | スピーカを鳴らす | CTRL + G |
| 8 | カーソルを1つ戻す | CTRL + H , DEL BS |
| 9 | タブレーション | CTRL + I , TAB |
| 10 | | CTRL + J |
| 11 | カーソルをホームポジションに戻す | CTRL + K |
| 12 | 画面をクリアし、カーソルをホームポジションに戻す | CTRL + L |
| 13 | 改行、一行入力 | CTRL + M , ↵ |
| 14 | | CTRL + N |
| 15 | | CTRL + O |

| コード | 機　　能 | 対応キー |
| --- | --- | --- |
| 16 | | CTRL + P |
| 17 | プログラムの実行や画面表示の一時停止解除 | CTRL + Q |
| 18 | 挿入モードに入いる | CTRL + R　PAST INS |
| 19 | プログラムの実行や画面表示の一時停止 | CTRL + S |
| 20 | | CTRL + T |
| 21 | 一行を画面から削除 | CTRL + U |
| 22 | | CTRL + V |
| 23 | | CTRL + W |
| 24 | | CTRL + X |
| 25 | | CTRL + Y |
| 26 | | CTRL + Z |
| 27 | エスケープシーケンスに入いる | ESC |
| 28 | カーソルを右へ移動 | → |
| 29 | カーソルを左へ移動 | ← |
| 30 | カーソルを上へ移動 | ↑ |
| 31 | カーソルを下へ移動 | ↓ |

## 6.7 キャラクタコード表

| 10進 | キャラクタ | 10進 | キャラクタ | 10進 | キャラクタ | 10進 | キャラクタ |
|---|---|---|---|---|---|---|---|
| 0 | コントロールコード表参照（文字として出力されない特殊コード） | 32 | | 64 | @ | 96 | ` |
| 1 | | 33 | ! | 65 | A | 97 | a |
| 2 | | 34 | " | 66 | B | 98 | b |
| 3 | | 35 | # | 67 | C | 99 | c |
| 4 | | 36 | $ | 68 | D | 100 | d |
| 5 | | 37 | % | 69 | E | 101 | e |
| 6 | | 38 | & | 70 | F | 102 | f |
| 7 | | 39 | ' | 71 | G | 103 | g |
| 8 | | 40 | ( | 72 | H | 104 | h |
| 9 | | 41 | ) | 73 | I | 105 | i |
| 10 | | 42 | * | 74 | J | 106 | j |
| 11 | | 43 | + | 75 | K | 107 | k |
| 12 | | 44 | , | 76 | L | 108 | l |
| 13 | | 45 | − | 77 | M | 109 | m |
| 14 | | 46 | . | 78 | N | 110 | n |
| 15 | | 47 | / | 79 | O | 111 | o |
| 16 | | 48 | 0 | 80 | P | 112 | p |
| 17 | | 49 | 1 | 81 | Q | 113 | q |
| 18 | | 50 | 2 | 82 | R | 114 | r |
| 19 | | 51 | 3 | 83 | S | 115 | s |
| 20 | | 52 | 4 | 84 | T | 116 | t |
| 21 | | 53 | 5 | 85 | U | 117 | u |
| 22 | | 54 | 6 | 86 | V | 118 | v |
| 23 | | 55 | 7 | 87 | W | 119 | w |
| 24 | | 56 | 8 | 88 | X | 120 | x |
| 25 | | 57 | 9 | 89 | Y | 121 | y |
| 26 | | 58 | : | 90 | Z | 122 | z |
| 27 | | 59 | ; | 91 | [ | 123 | { |
| 28 | | 60 | < | 92 | ¥ | 124 | ¦ |
| 29 | | 61 | = | 93 | ] | 125 | } |
| 30 | | 62 | > | 94 | ^ | 126 | ~ |
| 31 | | 63 | ? | 95 | _ | 127 | |

| 10進 | キャラクタ | 10進 | キャラクタ | 10進 | キャラクタ | 10進 | キャラクタ |
|---|---|---|---|---|---|---|---|
| 128 | ◀ | 160 | | 192 | タ | 224 | ┐ |
| 129 | ↵ | 161 | 。 | 193 | チ | 225 | ● |
| 130 | ▩ | 162 | 「 | 194 | ツ | 226 | ▲ |
| 131 | ═ | 163 | 」 | 195 | テ | 227 | |
| 132 | ‖ | 164 | 、 | 196 | ト | 228 | |
| 133 | ‖ | 165 | ・ | 197 | ナ | 229 | |
| 134 | ▏ | 166 | ヲ | 198 | ニ | 230 | |
| 135 | ▕ | 167 | ァ | 199 | ヌ | 231 | |
| 136 | ‖ | 168 | ィ | 200 | ネ | 232 | （ユーザ定義キャラクタ（CHS$関数を使って出力する）） |
| 137 | （ユーザ定義キャラクタ（キーボードから入力可能）） | 169 | ゥ | 201 | ノ | 233 | |
| 138 | | 170 | ェ | 202 | ハ | 234 | |
| 139 | | 171 | ォ | 203 | ヒ | 235 | |
| 140 | | 172 | ャ | 204 | フ | 236 | |
| 141 | | 173 | ュ | 205 | ヘ | 237 | |
| 142 | | 174 | ョ | 206 | ホ | 238 | |
| 143 | | 175 | ッ | 207 | マ | 239 | |
| 144 | | 176 | ー | 208 | ミ | 240 | |
| 145 | | 177 | ア | 209 | ム | 241 | |
| 146 | | 178 | イ | 210 | メ | 242 | |
| 147 | | 179 | ウ | 211 | モ | 243 | |
| 148 | | 180 | エ | 212 | ヤ | 244 | |
| 149 | | 181 | オ | 213 | ユ | 245 | |
| 150 | | 182 | カ | 214 | ヨ | 246 | |
| 151 | | 183 | キ | 215 | ラ | 247 | |
| 152 | | 184 | ク | 216 | リ | 248 | |
| 153 | | 185 | ケ | 217 | ル | 249 | |
| 154 | | 186 | コ | 218 | レ | 250 | |
| 155 | | 187 | サ | 219 | ロ | 251 | |
| 156 | | 188 | シ | 220 | ワ | 252 | |
| 157 | | 189 | ス | 221 | ン | 253 | |
| 158 | | 190 | セ | 222 | ゛ | 254 | |
| 159 | | 191 | ソ | 223 | ゜ | 255 | |

## ■エスケープシーケンス

BASIC 上で扱えるエスケープシーケンスは，TELCOM モードと共通ですので，詳しくはユーザーズマニュアルの第5章3節エスケープシーケンスを参照してください．ただし，カーソルの表示 "P" 及び "Q"）に関しては，プログラム実行中またはコマンド実行中のみ有効になります．

| ESC＋ | キャラクタコード | 機　　能 |
|---|---|---|
| E | 27，69 | 画面クリア |
| j | 27，106 | 画面クリア |
| K | 27，75 | カーソル位置から行末までの文字を消去 |
| J | 27，74 | カーソル位置から画面の終りまでの文字を消去 |
| l | 27，108 | カーソルのある行の文字を消去 |
| L | 27，76 | 1行挿入 |
| M | 27，77 | カーソルのある行を削除 |
| Y 〈y〉〈x〉 | | カーソルを指定位置へ移動* |
| A | 27，65 | カーソルを1行上へ移動 |
| B | 27，66 | カーソルを1行下へ移動 |
| C | 27，67 | カーソルを1文字右へ移動 |
| D | 27，68 | カーソルを1文字左へ移動 |
| E | 27，69 | カーソルを画面左上隅（ホームポジション）へ移動 |
| p | 27，112 | 白黒反転文字にする |
| q | 27，113 | 文字を正常にもどす |
| T | 27，84 | ファンクションキー表示を行う |
| U | 27，85 | ファンクションキー表示を消す |
| V | 27，86 | スクロール禁止（画面を固定） |
| W | 27，87 | スクロール許可 |
| P | 27，80 | カーソルを表示する |
| Q | 27，81 | カーソルを表示しない |

* ESC＋Y 〈y〉〈x〉

カーソル位置は，ESC＋Y に続く 2 文字によって垂直位置，水平位置の順で指定します．指定にはキャラクタコード 32 より大きい文字を使用して，空白(32)が 0 位置に相当し，以下順に〝！〟が 1，(〃) が 2 というようになります．例えば，ホームポジションにカーソルを移動するには，

ESC，〝Y〟，〝 〟，〝 〟

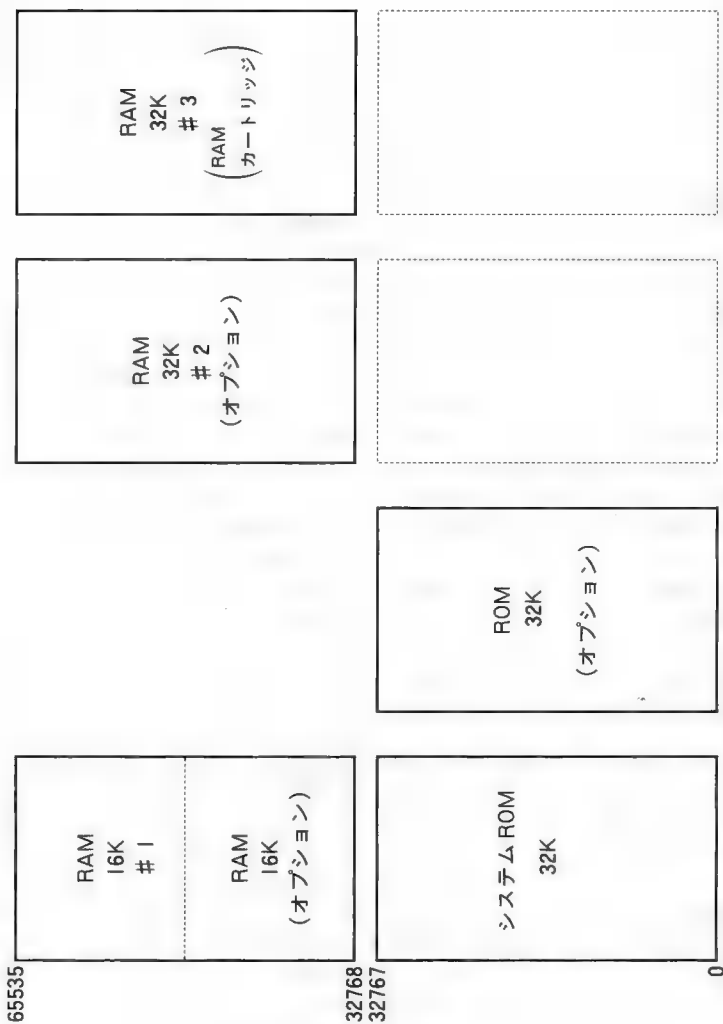
（キャラクタコードでは 27，89，32，32）

という順の文字列を送ればよいことになります．

■メモリマップ

1

| アドレス | 領域 | 備考 |
|---|---|---|
| 65535 | ワーク領域 | |
| 62336 | (Disk BASICコード) | |
| | ファイルコントロールブロック | MAXFILESにより変化 |
| | 文字列領域 | CLEAR文第1パラメータにより変化 |
| | FOR/GOSUB用スタック | |
| | システムスタック | |
| | | |
| | 配列領域 | |
| | 単純変数領域 | |
| | 機械語プログラムファイル .CO | |
| | ASCIIコードテキストファイル .DO | |
| 8000 | BASICプログラムファイル .BA | |

## 2

＊RAM＃2，RAM＃3のアドレスは0～32767または32768～65535のどちらにも指定できる．
＊各ブロックは，32kbyte単位でバンク切換を行うことができる．

## ■索引

## バンク2，バンク3のコールドスタートについて

メモリを増設し，バンク2（またはバンク3）を使う際のコールドスタートは，MENU モードの BANK コマンドを使います．

バンクを切り替える際には BANK コマンド（ f・10 キー）を使いますが，この時，f・10 キー（ SHIFT ＋ f・5 ）を押した直後に CTRL キーを押すことによって，切り替わったバンクがクリアされます．

例　バンク2のコールドスタート

① BANKコマンドによって，バンクを1に切り替える．
② SHIFT キーを押しながら，f・5 キーを押す．
③ f・5 キーを押した直後に CTRL キーを押す．

以上の操作でバンク2がコールドスタートします．バンク3のコールドスタートは，バンク2から，上記②，③の操作を行います．